X/Open CAE Specification

X/Open Common Desktop Environment (XCDE)

Services and Applications

X/Open Company Ltd.

X/Open CAE Specification

X/Open Common Desktop Environment (XCDE) Services and Applications

ISBN: 1-85912-074-1

X/Open Document Number: C323

Published by X/Open Company Ltd., U.K.

Any comments relating to the material contained in this document may be submitted to X/Open at:

X/Open Company Limited
Apex Plaza
Forbury Road
Reading
Berkshire, RG1 1AX
United Kingdom

or by Electronic Mail to:

XoSpecs@xopen.co.uk

Contents

Preface

X/Open

X/Open is an independent, worldwide, open systems organisation supported by most of the world's largest information systems suppliers, user organisations and software companies. Its mission is to bring to users greater value from computing, through the practical implementation of open systems.

X/Open's strategy for achieving this goal is to combine existing and emerging standards into a comprehensive, integrated, high-value and usable open system environment, called the Common Applications Environment (CAE). This environment covers the standards, above the hardware level, that are needed to support open systems. It provides for portability and interoperability of applications, and so protects investment in existing software while enabling additions and enhancements. It also allows users to move between systems with a minimum of retraining.

X/Open defines this CAE in a set of specifications which include an evolving portfolio of application programming interfaces (APIs) which significantly enhance portability of application programs at the source code level, along with definitions of and references to protocols and protocol profiles which significantly enhance the interoperability of applications and systems.

The X/Open CAE is implemented in real products and recognised by a distinctive trade mark — the X/Open brand — that is licensed by X/Open and may be used on products which have demonstrated their conformance.

X/Open Technical Publications

X/Open publishes a wide range of technical literature, the main part of which is focussed on specification development, but which also includes Guides, Snapshots, Technical Studies, Branding/Testing documents, industry surveys, and business titles.

There are two types of X/Open specification:

- *CAE Specifications*

  CAE (Common Applications Environment) specifications are the stable specifications that form the basis for X/Open-branded products. These specifications are intended to be used widely within the industry for product development and procurement purposes.

  Anyone developing products that implement an X/Open CAE specification can enjoy the benefits of a single, widely supported standard. In addition, they can demonstrate compliance with the majority of X/Open CAE specifications once these specifications are referenced in an X/Open component or profile definition and included in the X/Open branding programme.

  CAE specifications are published as soon as they are developed, not published to coincide with the launch of a particular X/Open brand. By making its specifications available in this way, X/Open makes it possible for conformant products to be developed as soon as is practicable, so enhancing the value of the X/Open brand as a procurement aid to users.

- *Preliminary Specifications*

  These specifications, which often address an emerging area of technology and consequently are not yet supported by multiple sources of stable conformant implementations, are released in a controlled manner for the purpose of validation through implementation of products. A Preliminary specification is not a draft specification. In fact, it is as stable as X/Open can make it, and on publication has gone through the same rigorous X/Open development and review procedures as a CAE specification.

  Preliminary specifications are analogous to the *trial-use* standards issued by formal standards organisations, and product development teams are encouraged to develop products on the basis of them. However, because of the nature of the technology that a Preliminary specification is addressing, it may be untried in multiple independent implementations, and may therefore change before being published as a CAE specification. There is always the intent to progress to a corresponding CAE specification, but the ability to do so depends on consensus among X/Open members. In all cases, any resulting CAE specification is made as upwards-compatible as possible. However, complete upwards-compatibility from the Preliminary to the CAE specification cannot be guaranteed.

In addition, X/Open publishes:

- *Guides*

  These provide information that X/Open believes is useful in the evaluation, procurement, development or management of open systems, particularly those that are X/Open-compliant. X/Open Guides are advisory, not normative, and should not be referenced for purposes of specifying or claiming X/Open conformance.

- *Technical Studies*

  X/Open Technical Studies present results of analyses performed by X/Open on subjects of interest in areas relevant to X/Open's Technical Programme. They are intended to communicate the findings to the outside world and, where appropriate, stimulate discussion and actions by other bodies and the industry in general.

- *Snapshots*

  These provide a mechanism for X/Open to disseminate information on its current direction and thinking, in advance of possible development of a Specification, Guide or Technical Study. The intention is to stimulate industry debate and prototyping, and solicit feedback. A Snapshot represents the interim results of an X/Open technical activity. Although at the time of its publication, there may be an intention to progress the activity towards publication of a Specification, Guide or Technical Study, X/Open is a consensus organisation, and makes no commitment regarding future development and further publication. Similarly, a Snapshot does not represent any commitment by X/Open members to develop any specific products.

Versions and Issues of Specifications

As with all *live* documents, CAE Specifications require revision, in this case as the subject technology develops and to align with emerging associated international standards. X/Open makes a distinction between revised specifications which are fully backward compatible and those which are not:

- a new *Version* indicates that this publication includes all the same (unchanged) definitive information from the previous publication of that title, but also includes extensions or additional information. As such, it *replaces* the previous publication.

- a new *Issue* does include changes to the definitive information contained in the previous publication of that title (and may also include extensions or additional information). As such, X/Open maintains *both* the previous and new issue as current publications.

Corrigenda

Most X/Open publications deal with technology at the leading edge of open systems development. Feedback from implementation experience gained from using these publications occasionally uncovers errors or inconsistencies. Significant errors or recommended solutions to reported problems are communicated by means of Corrigenda.

The reader of this document is advised to check periodically if any Corrigenda apply to this publication. This may be done either by email to the X/Open info-server or by checking the Corrigenda list in the latest X/Open Publications Price List.

To request Corrigenda information by email, send a message to info-server@xopen.co.uk with the following in the Subject line:

```
request corrigenda; topic index
```

This will return the index of publications for which Corrigenda exist.

This Document

There are two X/Open CAE Specifications (see above) defining the X/Open Common Desktop Environment (XCDE) requirements:

- X/Open Common Desktop Environment — Definitions and Infrastructure (**XCDI**)
- X/Open Common Desktop Environment — Services and Applications (**XCSA**) (this document)

The **XCDI** and **XCSA** documents are mutually dependent specifications, which have been split into two volumes for convenience of use and publication.

The **XCDI** specification provides common definitions for the **XCDI** specification and the **XCSA** specification; therefore, readers should be familiar with the **XCDI** specification before using the **XCSA** specification. (Readers are also expected to be familiar with the X/Open CAE Specification, **System Interface Definitions**, **Issue 4, Version 2**, which contains a number of applicable definitions.)

Structure

The **XCSA** specification is structured as follows:

- Chapter 1 describes the XCDE window management services.
- Chapter 2 describes the XCDE workspace management services.
- Chapter 3 describes the XCDE session management services.
- Chapter 4 describes the XCDE help services.
- Chapter 5 describes the XCDE calendar and appointment services.
- Chapter 6 describes the XCDE mail services.
- Chapter 7 describes the XCDE file management services.
- Chapter 8 describes the XCDE front panel services.
- Chapter 9 describes the XCDE text editing services.

- Chapter 10 describes the XCDE icon editing services.
- Chapter 11 describes the XCDE GUI scripting (windowing KornShell) services.
- Chapter 12 describes the XCDE terminal emulation services.
- Chapter 13 describes the XCDE style management services.
- Chapter 14 describes the XCDE application building services.
- Chapter 15 describes the XCDE application integration services.
- Chapter 16 describes the XCDE action creation services.
- Chapter 17 describes the XCDE print queue services.
- Chapter 18 describes the XCDE calculator services.
- Chapter 19 describes font and icon conventions for XCDE applications.
- Chapter 20 describes the style requirements for XCDE applications.

Comprehensive references are available in the index.

Typographical Conventions

The following typographical conventions are used throughout this document:

- **Bold** font is used in text for options to commands, filenames, keywords and type names. It is also used to identify brackets that surround optional items in syntax, **[ ]**.
- *Italic* strings are used for emphasis or to identify the first instance of a word requiring definition. Italics in text also denote:
  - — variable names, for example, substitutable argument prototypes
  - — environment variables, which are also shown in capitals
  - — commands or utilities
  - — external variables, such as *errno*
  - — X Window System widgets
  - — functions; these are shown as follows: *name*( ); names without parentheses are either external variables or function family names
- Normal font is used for the names of constants and literals.
- The notation **<file.h>** indicates a header file.
- Names surrounded by braces, for example, {ARG_MAX}, represent symbolic limits or configuration values which may be declared in appropriate headers by means of the C **#define** construct.
- Ellipses, . . ., are used to show that additional arguments are optional.
- Syntax and code examples are shown in `fixed width font`. Brackets shown in this font, `[ ]`, are part of the syntax and do *not* indicate optional items.
- Variables within syntax statements are shown in *`italic fixed width font`*.
- The names of virtual keys, such as <*Help*> or <*Insert*> are used as described by the model keyboard section of the **OSF/Motif Style Guide**.

Trade Marks

DEC® is a registered trade mark of Digital Equipment Corporation.

Helvetica® is a registered trade mark of Linotype AG and/or its subsidiaries.

IBM® is a registered trade mark of International Business Machines Corporation.

Motif™ is a trade mark of Open Software Foundation, Inc.

OPEN LOOK® is a registered trademark of Novell, Inc.

Postscript® is a registered trade mark of Adobe Systems Incorporated.

ToolTalk™ is a trade mark of Sun Microsystems, Inc.

UNIX® is a registered trade mark in the United States and other countries, licensed exclusively through X/Open Company Limited.

X/Open® is a registered trade mark, and the ‘‘X’’ device is a trade mark, of X/Open Company Ltd.

X Window System™ is a trade mark of the Massachusetts Institute of Technology.

Acknowledgements

X/Open gratefully acknowledges the CDE sponsoring companies who donated the materials for this specification:

- Hewlett-Packard Company
- International Business Machines Corporation
- Novell, Incorporated.
- Sun Microsystems, Incorporated

X/Open also acknowledges the XAPI Association (XAPIA) for their contribution to the **Calendar and Appointment Services** specification referred to in Chapter 5 of this document. See also the referenced X/Open CAE Specification, **Calendaring and Scheduling API (XCS)**.

Referenced Documents

The following documents are referenced in this specification:

ISO C
: ISO / IEC 9899: 1990, Information technology — Programming Languages — C.

ISO / IEC 6429: 1992
: Information processing — ISO 7-bit and 8-bit coded character sets — Control functions for coded character sets

ISO 8859-1: 1987
: Information processing — 8-bit single-byte coded graphic character sets — Part 1: Latin alphabet No. 1

ISO 8879: 1986
: Information processing — Text and office systems — Standard Generalised Markup Language (SGML)

ISO / IEC 9070: 1991
: Information technology — SGML support facilities — Registration procedures for public text owner identifiers

ANSI X3.64-1979
: Additional Controls for Use with the American National Standard Code for Information Interchange

RFC-822
: Internet RFC 822, Crocker, D. Standard for the format of ARPA Internet text messages.

MIME RFCs
: Internet RFC 1521, N. Borenstein, N. Freed, MIME (Multipurpose Internet Mail Extensions) Part One: Mechanisms for Specifying and Describing the Format of Internet Message Bodies.

  Internet RFC 1522, K. Moore, MIME (Multipurpose Internet Mail Extensions) Part Two: Message Header Extensions for Non-ASCII Text.

  Internet RFC 1590, J. Postel, Media Type Registration Procedure.

Motif Style Guide
: Open Software Foundation, OSF / Motif Style Guide, Revision 1.2 (ISBN: 0-13-643123-2).

The following X / Open documents are referenced in this specification.

XBD
: X / Open CAE Specification, September 1994, System Interface Definitions, Issue 4, Version 2 (ISBN: 1-859120-36-9, C434).

XSH
: X / Open CAE Specification, September 1994, System Interfaces and Headers, Issue 4, Version 2, September 1994 (ISBN: 1-859120-37-7, C435).

XCU
: X / Open CAE Specification, Commands and Utilities, Issue 4, Version 2, September 1994 (ISBN: 1-859120-34-2, C436).

XIG
: X/Open Guide, Internationalisation Guide, July 1993 (ISBN: 1-85912-002-4, G304).

XNFS
: X/Open CAE Specification, Protocols for X/Open Interworking, September 1992, (ISBN: i-872630-66-9, C218). This includes description of XDR (Sun Microsystems' External Data Representation standard), which was originally described in Internet RFC 1014.

XPG4
: X/Open Single UNIX Specification (Spec. 1170) — Four Volume Set, September 1994 (ISBN: 1-85912-054-7, T405).

X Protocol
: X/Open CAE Specification, Window Management (X11R5): X Window System Protocol, April 1995 (ISBN: 1-85912-087-3, C507).

Xlib
: X/Open CAE Specification, Window Management (X11R5): Xlib — C Language Binding, April 1995 (ISBN: 1-85912-088-1, C508).

Xt
: X/Open CAE Specification, Window Management (X11R5): X Toolkit Intrinsics, April 1995 (ISBN: 1-85912-089-X, C509).

ICCCM
: X/Open CAE Specification, Window Management (X11R5): File Formats and Application Conventions, April 1995 (ISBN: 1-85912-090-3, C510).

Motif
: X/Open CAE Specification, Motif Toolkit API, March 1995 (ISBN: 1-85912-024-5, C320).

XCS
: X/Open CAE Specification, Calendaring and Scheduling API (XCS), March 1995 (ISBN: 1-85912-076-8, C321).

XCDI
: X/Open CAE Specification, Common Desktop Environment: Definitions and Infrastructure (XCDI), March 1995 (ISBN: 1-85912-070-9, C324).

Chapter 1

Window Management Services

1.1 Introduction

This chapter describes the XCDE window management services. The XCDE window manager is a superset of the X/Open Motif *mwm* window manager, which includes the following major enhancements:

- Support for historical OPEN LOOK applications. (OPEN LOOK hints are mapped to the nearest Motif behaviour.)
- Support for multiple workspaces. See Chapter 2 on page 3.
- Support for the front panel. See Chapter 8 on page 113.

Many features of the window manager can be configured via X resources and configuration files, including:

- Root window pop-up menu.
- Window menu for client windows.
- Key and button bindings for raising and lowering client windows.

Commonly customised window manager features, such as the focus policy, can also be configured through dialogs in the style manager.

The window manager provides API support for the following tasks:

- Standard ICCCM and Motif capabilities (for example, iconify a window). These functions are described in the following standards:
  - X/Open CAE Specification, **Window Management: File Formats and Application Conventions**
  - X/Open CAE Specification, **Window Management: Xlib C Language Binding**
  - X/Open CAE Specification, **Motif Toolkit API**
- Selecting icons placed on the root window but controlled by other applications

There is one additional functional difference between the XCDE window manager and *mwm*:

- By default, the only keyboard accelerator included in the window menu for the X/Open Common Desktop Environment window manager is Close (Alt/F4).

1.2 Data Formats

The resource description file **$HOME/.dtwmrc** is equivalent in function to the **$HOME/.mwmrc** described in the X/Open CAE Specification, **Motif Toolkit API**, which provides customisation abilities for users of the Motif Window Manager. The **$HOME/.dtwmrc** file allows customisation for users of the XCDE Window Manager. It has identical formatting to **$HOME/.mwmrc**, except that the resource class name **Mwm** is replaced by the name **Dtwm**.

Chapter 2

Workspace Management Services

2.1 Introduction

The XCDE workspace manager provides support for multiple workspaces. Each workspace is a ‘‘virtual screen’’; windows can be placed in a single workspace, all workspaces or any combination of individual workspaces. The initial number of workspaces is determined by a resource when the workspace manager starts up. Workspaces may be added, deleted or renamed dynamically. Workspace switching can be done through the workspace switch in the front panel or by binding a workspace manager function to a button, key or window manager menu. The workspace manager cooperates with the session manager to save the workspace state between sessions.

The workspace manager provides API support for querying and controlling the workspace state and adding and removing windows from workspaces. Clients need not be aware of workspaces or the workspace manager to operate properly in the XCDE.

2.2 Functions

This section defines the functions, macros and external variables that provide XCDE workspace management services to support application portability at the C-language source level.

NAME

DtWsmAddCurrentWorkspaceCallback — add a callback to be called when the current workspace changes

SYNOPSIS

```
#include <Dt/Wsm.h>

DtWsmCBContext
DtWsmAddCurrentWorkspaceCallback(Widget widget,
                                 DtWsmWsChangeProc ws_change,
                                 Pointer client_data);
```

DESCRIPTION

The *DtWsmAddCurrentWorkspaceCallback*( ) function registers an application function to be called when the workspace manager switches to a new workspace.

The workspace manager sends the new current workspace name to the **DtWsmWsChangeProc** callback.

The *widget* argument is a realised widget.

The *ws_change* argument is the procedure to be called when the workspace changes.

The *client_data* argument points to arbitrary client data to be passed back to *ws_change*.

The header defines the **DtWsmWsChangeProc** callback prototype as follows:

```
typedef void (*DtWsmWsChangeProc)(Widget widget,
                                  Atom aWorkspace,
                                  Pointer client_data);
```

The *widget* argument is the ID of the widget to be registered with the callback.

The *aWorkspace* argument is the name of the new current workspace (converted to an X atom).

The *client_data* argument points to the client data to be registered with the callback.

RETURN VALUE

Upon successful completion, the *DtWsmAddCurrentWorkspaceCallback*( ) function returns a workspace callback registration context.

APPLICATION USAGE

The *DtWsmAddCurrentWorkspaceCallback*( ) function returns a registration context that the application must save in order to remove this callback later. *DtWsmAddCurrentWorkspaceCallback*( ) requires a window; thus, a gadget is not acceptable for the *widget* argument. The *DtWsmRemoveWorkspaceCallback*( ) function needs a registration context to remove the callback.

SEE ALSO

<Dt/Wsm.h>, *DtWsmRemoveWorkspaceCallback*( ).

CHANGE HISTORY

First released in Issue 1.

NAME

DtWsmAddWorkspaceFunctions — add workspace functions for a window

SYNOPSIS

```
#include <Dt/Wsm.h>

void DtWsmAddWorkspaceFunctions(Display *display,
                                Window window);
```

DESCRIPTION

The *DtWsmAddWorkspaceFunctions*( ) function enables workspace functions for a window. When workspace functions are enabled, the default window menu for the window displayed by the workspace manager shows entries that allow the window to occupy a different set of workspaces, occupy all workspaces, or be removed from the current workspace.

The *display* argument is the X display.

The *window* argument is the window to have its workspace functions enabled.

RETURN VALUE

The *DtWsmAddWorkspaceFunctions*( ) function returns no value.

APPLICATION USAGE

The application must call the *DtWsmAddWorkspaceFunctions*( ) function before mapping the window. The workspace manager only looks at the workspace function information at the time the workspace manager manages the window. If the workspace manager is currently managing the window, the application must withdraw the window (see *XWithdrawWindow*( )), add the workspace functions, and map the window once again.

By default, the workspace manager enables workspace functions.

The application should call *DtWsmAddWorkspaceFunctions*( ) to restore the workspace functions removed by the *DtWsmRemoveWorkspaceFunctions*( ) function.

SEE ALSO

<Dt/Wsm.h>, *XWithdrawWindow*( ) in the X / Open CAE Specification, **Window Management: Xlib C Language Binding**; *DtWsmRemoveWorkspaceFunctions*( ).

CHANGE HISTORY

First released in Issue 1.

NAME

DtWsmAddWorkspaceModifiedCallback — add a callback to be called when any workspace is changed

SYNOPSIS

```
#include <Dt/Wsm.h>

DtWsmCBContext
DtWsmAddWorkspaceModifiedCallback(Widget widget,
                                  DtWsmWsModifiedProc ws_change,
                                  Pointer client_data);
```

DESCRIPTION

The *DtWsmAddWorkspaceModifiedCallback*( ) function works with the workspace manager and registers a function to be called when a workspace is added, deleted or modified.

The *widget* argument is a realised widget.

The *ws_change* argument is the procedure to be called when a workspace is modified.

The *client_data* argument points to arbitrary client data to be passed back to *ws_change*.

The header defines the **DtWsmWsModifiedProc** callback prototype as follows:

```
typedef void (*DtWsmWsModifiedProc)(Widget widget,
                                    Atom aWorkspace,
                                    DtWsmWsReason reason,
                                    Pointer client_data);
```

The *widget* argument is the ID of the widget to be registered with the callback.

The *aWorkspace* argument is the name of the new current workspace (converted to an X atom).

The *reason* argument is a type of modification:

DtWSM_REASON_ADD
: A new workspace was added.

DtWSM_REASON_BACKDROP
: The backdrop for the workspace changed.

DtWSM_REASON_CURRENT
: A different workspace was made the current workspace.

DtWSM_REASON_DELETE
: A workspace was deleted.

DtWSM_REASON_TITLE
: The workspace title changed.

The *client_data* argument points to the client data to be registered with the callback.

RETURN VALUE

Upon successful completion, the *DtWsmAddWorkspaceModifiedCallback*( ) function returns a workspace callback registration context.

APPLICATION USAGE

The *DtWsmAddWorkspaceModifiedCallback*( ) function returns a registration context that the application must save in order to remove this callback later.

DtWsmAddWorkspaceModifiedCallback( ) requires a window; thus, a gadget is not acceptable for the *widget* argument.

The *DtWsmRemoveWorkspaceCallback*( ) function needs a registration context to remove the callback.

SEE ALSO

<Dt/Wsm.h>, *DtWsmGetWorkspaceInfo*( ), *DtWsmGetWorkspaceList*( ), *DtWsmRemoveWorkspaceCallback*( ).

CHANGE HISTORY

First released in Issue 1.

NAME

DtWsmFreeWorkspaceInfo — free workspace information

SYNOPSIS

```
#include <Dt/Wsm.h>

void DtWsmFreeWorkspaceInfo(DtWsmWorkspaceInfo *pWsInfo);
```

DESCRIPTION

The *DtWsmFreeWorkspaceInfo*( ) function frees workspace information.

The *pWsInfo* argument points to the workspace information the *DtWsmGetWorkspaceInfo*( ) function returns.

RETURN VALUE

The *DtWsmFreeWorkspaceInfo*( ) function returns no value.

APPLICATION USAGE

The data space for **DtWsmWorkspaceInfo** is allocated by *DtWsmGetWorkspaceInfo*( ). The application must call *DtWsmFreeWorkspaceInfo*( ) to free the data.

SEE ALSO

<Dt/Wsm.h>, *DtWsmGetWorkspaceInfo*( ).

CHANGE HISTORY

First released in Issue 1.

NAME

DtWsmGetCurrentBackdropWindow — get the backdrop window for the current workspace

SYNOPSIS

```
#include <Dt/Wsm.h>

Window DtWsmGetCurrentBackdropWindow(Display *display,
                                     Window root);
```

DESCRIPTION

The *DtWsmGetCurrentBackdropWindow*( ) function works with the workspace manager and returns the window used as the backdrop for the current workspace.

The *display* argument is the X display.

The *root* argument is the root window of the screen of interest.

RETURN VALUE

Upon successful completion, the *DtWsmGetCurrentBackdropWindow*( ) returns the window used as the backdrop for the current workspace. The function returns None if there is no backdrop window for the workspace or if the workspace manager is not running.

APPLICATION USAGE

If the *DtWsmGetCurrentBackdropWindow*( ) function is not successful, failure may be due to a memory allocation error or failure to find the correct workspace information (that is, the workspace manager is not running).

SEE ALSO

<Dt/Wsm.h>, *DtWsmGetCurrentWorkspace*( ), *DtWsmGetWorkspaceInfo*( ).

CHANGE HISTORY

First released in Issue 1.

NAME

DtWsmGetCurrentWorkspace — get the current workspace

SYNOPSIS

```
#include <Dt/Wsm.h>

Status DtWsmGetCurrentWorkspace(Display *display,
                                Window root,
                                Atom *paWorkspace);
```

DESCRIPTION

The *DtWsmGetCurrentWorkspace*( ) function works with the workspace manager and returns the name of the current workspace (converted to an X atom).

The *display* argument is the X display.

The *root* argument is the root window of the screen of interest.

The *paWorkspace* argument is the address of an atom to receive the current workspace identifier.

RETURN VALUE

Upon successful completion, the *DtWsmGetCurrentWorkspace*( ) function returns Success and the atom identifying the current workspace is returned in *paWorkspace*; otherwise, it returns a value not equal to Success.

APPLICATION USAGE

If the *DtWsmGetCurrentWorkspace*( ) function is not successful, the most likely reason for failure is that the workspace manager is not running.

SEE ALSO

<Dt/Wsm.h>.

CHANGE HISTORY

First released in Issue 1.

NAME

DtWsmGetWorkspaceInfo — get detailed workspace information

SYNOPSIS

```
#include <Dt/Wsm.h>

Status DtWsmGetWorkspaceInfo(Display *display,
                             Window root,
                             Atom aWorkspace,
                             DtWsmWorkspaceInfo **ppWsInfo);
```

DESCRIPTION

The *DtWsmGetWorkspaceInfo*( ) function works with the workspace manager and returns detailed information on a specific workspace.

The *display* argument is the X display.

The *root* argument is the root window of the screen of interest.

The *aWorkspace* argument is the workspace name (converted to an X atom).

The **ppWsInfo* argument is the address of a variable to receive the returned pointer to the workspace information data.

The **DtWsmWorkspaceInfo** structure contains at least the following members:

`Atom`	`workspace`	The workspace name (converted to an X atom).
`unsigned long`	`bg`	The pixel ID used for the background colour of the backdrop.
`unsigned long`	`fg`	The pixel ID used for the foreground colour of the backdrop.
`Atom`	`backdropName`	The backdrop file name (converted to an X atom). The file must be in either X Bitmap file format (with extension **.bm**) or X Pixmap file format (with extension **.pm**). The workspace management services look for the file along the same path used for searching icons. The directory **/usr/dt/backdrops** is the default directory if the file cannot be found along the icon search path.
`int`	`colorSetId`	The colourset number used for this workspace, which affects the backdrop colour and the button colour for this workspace on the front panel.
`char`	`*pchTitle`	The title displayed in the button for this workspace on the front panel. This string is interpreted in the locale in which the workspace manager is running. The title is different from the workspace name. The workspace name, when converted from an X atom, is used as the identifier for a workspace in the workspace manager function calls. The workspace manager also uses the workspace name as a resource name; thus, the characters used in a

		workspace name are restricted to the characters in the X Portable Character Set. The workspace name for a workspace created from the front panel is generated automatically by the workspace manager.
`Window`	`*backdropWindows`	A pointer to an array of windows that make up the backdrop.
`int`	`numBackdropWindows`	The number of elements in the *backdropWindows* array.

RETURN VALUE

Upon successful completion, the *DtWsmGetWorkspaceInfo*( ) function returns Success and the workspace manager returns in **ppWsInfo* a pointer to a **DtWsmInfo** structure that contains information about the workspace *aWorkspace*; otherwise, it returns a value not equal to Success.

APPLICATION USAGE

If the *DtWsmGetWorkspaceInfo*( ) function is not successful, failure may be due to a memory allocation error or failure to find the correct workspace information (that is, the workspace manager is not running). The application must use the *DtWsmFreeWorkspaceInfo*( ) function to free the data returned in **ppWsInfo*. The **backdropWindows* pointer may be useful for applications that are interested in some events on the root window. Since the backdrop covers the root window, the backdrop catches the button events before they reach the root.

SEE ALSO

<Dt/Wsm.h>, *DtWsmGetWorkspaceList*( ), *DtWsmFreeWorkspaceInfo*( ).

CHANGE HISTORY

First released in Issue 1.

NAME

DtWsmGetWorkspaceList — get the names of the currently defined workspaces

SYNOPSIS

```
#include <Dt/Wsm.h>

Status DtWsmGetWorkspaceList(Display *display,
                             Window root,
                             Atom **ppaWorkspaces,
                             int *pNumWorkspaces);
```

DESCRIPTION

The *DtWsmGetWorkspaceList*( ) function works with the workspace manager and returns a list of the names (converted into X atoms) of the currently defined workspaces.

The *display* argument is the X display.

The *root* argument is the root window of the screen of interest.

The **ppaWorkspaces* argument is the address of a pointer to receive the returned pointer to the workspacelist.

The *pNumWorkspaces* argument is the address of an integer to receive the number of elements in the workspacelist.

RETURN VALUE

Upon successful completion, the *DtWsmGetWorkspaceList*( ) function returns Success and the workspace manager returns in **ppaWorkspaces* the list of atoms identifying workspaces, and returns in *pNumWorkspaces* the number of workspaces.

APPLICATION USAGE

If the *DtWsmGetWorkspaceList*( ) function is not successful, failure may be due to a memory allocation error or failure to find the correct workspace information (that is, the workspace manager is not running). To get detailed information on the workspaces, the application must first call the *DtWsmGetWorkspaceList*( ) function to get the names of all the workspaces. Then, for each workspace in the list, the application must call the *DtWsmGetWorkspaceInfo*( ) function. The application must use *XtFree*( ) to free data returned in **ppaWorkspaces*.

SEE ALSO

<Dt/Wsm.h>, *XtFree*( ) in the X/Open CAE Specification, **Window Management: X Toolkit Intrinsics**; *DtWsmGetWorkspaceInfo*( ).

CHANGE HISTORY

First released in Issue 1.

NAME

DtWsmGetWorkspacesOccupied — get the workspaces in which a window resides

SYNOPSIS

```
#include <Dt/Wsm.h>

Status DtWsmGetWorkspacesOccupied(Display *display,
                                  Window window,
                                  Atom **ppaWorkspaces,
                                  int *pNumWs);
```

DESCRIPTION

The *DtWsmGetWorkspacesOccupied*( ) function works with the workspace manager and returns the list of workspaces in which this window resides. Each element of the list is the name of a workspace (converted to an X atom).

The *display* argument is the X display.

The *window* argument is the window of interest.

The **ppaWorkspaces* argument is the address of a pointer to receive the pointer to a list of workspace names (converted to X atoms).

The *pNumWs* argument is the address of an integer to receive the number of elements in the list of workspaces returned in **ppaWorkspaces*.

RETURN VALUE

Upon successful completion, the *DtWsmGetWorkspacesOccupied*( ) function returns Success and the workspace manager returns in **ppaWorkspaces* a list of atoms identifying the occupied workspaces, and returns in *pNumWs* the number of occupied workspaces; otherwise, it returns a value not equal to Success.

APPLICATION USAGE

If the *DtWsmGetWorkspacesOccupied*( ) function is not successful, failure may be due to a memory allocation error or failure to find the correct workspace information (that is, the workspace manager is not running). The application must use *XtFree*( ) to free data returned in **ppaWorkspaces*.

SEE ALSO

<Dt/Wsm.h>, *XtFree*( ) in the X/Open CAE Specification, **Window Management: X Toolkit Intrinsics**.

CHANGE HISTORY

First released in Issue 1.

NAME

DtWsmOccupyAllWorkspaces — put a window into all workspaces

SYNOPSIS

```
#include <Dt/Wsm.h>

void DtWsmOccupyAllWorkspaces(Display *display,
                              Window window);
```

DESCRIPTION

The *DtWsmOccupyAllWorkspaces*( ) function works with the workspace manager and puts a window into all currently defined workspaces and also into newly created workspaces.

The *display* argument is the X display.

The *window* argument is the window to occupy all workspaces.

Calling the *DtWsmSetWorkspacesOccupied*( ) function overrides the effect of the *DtWsmOccupyAllWorkspaces*( ) function.

RETURN VALUE

The *DtWsmOccupyAllWorkspaces*( ) function returns no value.

SEE ALSO

<Dt/Wsm.h>, *DtWsmSetWorkspacesOccupied*( ).

CHANGE HISTORY

First released in Issue 1.

NAME

DtWsmRemoveWorkspaceCallback — remove a workspace callback

SYNOPSIS

```
#include <Dt/Wsm.h>

void DtWsmRemoveWorkspaceCallback(DtWsmCBContext context);
```

DESCRIPTION

The *DtWsmRemoveWorkspaceCallback*( ) function works with the workspace manager and removes a callback called when the current workspace changes or when a workspace is modified.

The *context* argument is the context the *DtWsmAddWorkspaceCallback*( ) function or the *DtWsmAddWorkspaceModifiedCallback*( ) function returns when the application registers the callback.

RETURN VALUE

The *DtWsmRemoveWorkspaceCallback*( ) function returns no value.

SEE ALSO

<Dt/Wsm.h>, *DtWsmAddWorkspaceModifiedCallback*( ), *DtWsmAddCurrentWorkspaceCallback*( ).

CHANGE HISTORY

First released in Issue 1.

NAME

DtWsmRemoveWorkspaceFunctions — remove a window's workspace functions

SYNOPSIS

```
#include <Dt/Wsm.h>

void DtWsmRemoveWorkspaceFunctions(Display *display,
                                   Window window);
```

DESCRIPTION

The *DtWsmRemoveWorkspaceFunctions*( ) function removes a window's workspace functions. When *DtWsmRemoveWorkspaceFunctions*( ) removes workspace functions, the window menu for the window the workspace manager displays does not have the entries that allow the window to occupy a different set of workspaces, occupy all workspaces, or be removed from the current workspace.

The *display* argument is the X display.

The *window* argument is the window to have its workspace functions disabled.

RETURN VALUE

The *DtWsmRemoveWorkspaceFunctions*( ) function returns no value.

APPLICATION USAGE

The application must call *DtWsmRemoveWorkspaceFunctions*( ) before the window is mapped. The workspace manager only looks at the workspace function information at the time the workspace manager manages the window. If the workspace manager is managing the window, the application must withdraw the window (see *XWithdrawWindow*( )), remove the workspace functions, and map the window once again.

SEE ALSO

XWithdrawWindow( ) in the X/Open CAE Specification, **Window Management: Xlib C Language Binding**; **<Dt/Wsm.h>**.

CHANGE HISTORY

First released in Issue 1.

NAME

DtWsmSetCurrentWorkspace — set the current workspace

SYNOPSIS

```
#include <Dt/Wsm.h>

Status DtWsmSetCurrentWorkspace(Widget widget,
                                Atom aWorkspace);
```

DESCRIPTION

The *DtWsmSetCurrentWorkspace*( ) function works with the workspace manager and sets the current workspace. Applications can use this function to switch from the current workspace to another workspace.

The *widget* argument is a realised widget on the screen of interest.

The *aWorkspace* argument is the name (in X atom form) of the workspace to be made current.

RETURN VALUE

Upon successful completion, the *DtWsmSetCurrentWorkspace*( ) function returns Success; otherwise, it returns a value not equal to Success.

APPLICATION USAGE

If the *DtWsmSetCurrentWorkspace*( ) function is not successful, the most likely reason for failure is that the workspace manager is not running. The *DtWsmSetCurrentWorkspace*( ) function requires a widget. A gadget is not acceptable for the *widget* argument.

DtWsmSetCurrentWorkspace( ) sends a message to the workspace manager to switch workspaces. If the workspace name is not valid, no action is taken and the workspace manager reports no error.

SEE ALSO

<Dt/Wsm.h>, *DtWsmGetCurrentWorkspace*( ).

CHANGE HISTORY

First released in Issue 1.

NAME

DtWsmSetWorkspacesOccupied — set the workspaces in which a window resides

SYNOPSIS

```
#include <Dt/Wsm.h>

void DtWsmSetWorkspacesOccupied(Display *display,
                                Window window,
                                Atom *paWorkspaces,
                                int numWs);
```

DESCRIPTION

The *DtWsmSetWorkspacesOccupied*( ) function works with the workspace manager and puts a window into a set of workspaces.

The *display* argument is the X display.

The *window* argument is the window to be put into this set of workspaces.

The *paWorkspaces* argument points to a list of workspace names (converted to X atoms); the workspace manager places the window into these workspaces.

The *numWs* argument is the number of elements in the list of workspaces.

The *DtWsmSetWorkspacesOccupied*( ) function does not validate the list of requested workspaces. The workspace manager ignores invalid workspaces in the list.

Calling the *DtWsmSetWorkspacesOccupied*( ) function overrides the effect of the *DtWsmOccupyAllWorkspaces*( ) function.

RETURN VALUE

The *DtWsmSetWorkspacesOccupied*( ) function returns no value.

SEE ALSO

<Dt/Wsm.h>.

CHANGE HISTORY

First released in Issue 1.

2.3 Headers

This section describes the contents of headers used by the XCDE workspace management service functions, macros and external variables.

Headers contain the definition of symbolic constants, common structures, preprocessor macros and defined types. Each function in Section 2.2 specifies the headers that an application must include in order to use that function. In most cases only one header is required. These headers are present on an application development system; they do not have to be present on the target execution system.

NAME

Dt/Wsm.h — workspace manager definitions

SYNOPSIS

```
#include <Dt/Wsm.h>
```

DESCRIPTION

The **<Dt/Wsm.h>** header defines structures and function prototypes for workspace management services.

The **DtWsmWorkspaceInfo** structure contains at least the following members:

`Atom`	`workspace`	X atom name for the workspace
`unsigned long`	`bg`	Backdrop background pixel
`unsigned long`	`fg`	Backdrop foreground pixel
`Atom`	`backdropName`	X atom name for backdrop
`int`	`colorSetId`	Number of colour set used
`char`	`*pchTitle`	Title of workspace
`Window`	`backdropWindow`	The backdrop window for the current workspace
`int`	`numBackdropWindows`	The number of elements in the *backdropWindows* array.

The **DtWsmCBContext** structure is opaque. Workspace management functions that add callbacks to uniquely identify callback functions for later removal, return the **DtWsmCBContext** structure.

The header defines the following **DtWsmWsReason** constants:

```
DtWSM_REASON_ADD
DtWSM_REASON_DELETE
DtWSM_REASON_BACKDROP
DtWSM_REASON_TITLE
DtWSM_REASON_CURRENT
```

The header defines the following functions:

```
DtWsmCBContext
DtWsmAddCurrentWorkspaceCallback(Widget widget,
                                 DtWsmWsChangeProc ws_change,
                                 Pointer client_data);

void DtWsmAddWorkspaceFunctions(Display *display,
                                Window window);

void DtWsmRemoveWorkspaceFunctions(Display *display,
                                   Window window);

DtWsmCBContext
DtWsmAddWorkspaceModifiedCallback(Widget widget,
                                  DtWsmWsModifiedProc ws_change,
                                  Pointer client_data);

void DtWsmRemoveWorkspaceCallback(DtWsmCBContext context);

void DtWsmFreeWorkspaceInfo(DtWsmWorkspaceInfo *pWsInfo);
```

```
Status DtWsmGetCurrentBackdropWindows(Display *display,
                                      Window root);

Status DtWsmGetCurrentWorkspace(Display *display,
                                Window root,
                                Atom *paWorkspace);

Status DtWsmSetCurrentWorkspace(Widget widget,
                                Atom aWorkspace);

Status DtWsmGetWorkspaceInfo(Display *display,
                             Window root,
                             Atom aWorkspace,
                             DtWsmWorkspaceInfo **ppWsInfo);

Status DtWsmGetWorkspaceList(Display *display,
                             Window root,
                             Atom **ppaWorkspaces,
                             int *pNumWs);

Status DtWsmGetWorkspacesOccupied(Display *display,
                                  Window window,
                                  Atom **ppaWorkspace,
                                  int *pNumWs);

void DtWsmSetWorkspacesOccupied(Display *display,
                                Window window,
                                Atom *paWorkspaces,
                                int numWs);

void DtWsmOccupyAllWorkspaces(Display *display,
                              Window window);
```

CHANGE HISTORY

First released in Issue 1.

Chapter 3

Session Management Services

3.1 Introduction

The XCDE session management services provides ICCCM Version 1.1 session management during a user's session, from login to logout. These services allow for saving a session, restoring a session, locking a session and launching screen savers.

A session is the collection of applications, settings and resources that are present on the user's desktop. Session management is a set of conventions and protocols that allow a session manager to save and restore a user's session.

3.2 Functions

This section defines the functions, macros and external variables that provide XCDE session management services to support application portability at the C-language source level.

NAME

DtSaverGetWindows — get the list of windows for drawing by a screen saver application

SYNOPSIS

```
#include <Dt/Saver.h>

Boolean DtSaverGetWindows(Display *display,
                          Window **window,
                          int *count);
```

DESCRIPTION

The *DtSaverGetWindows*( ) function returns a list of windows on which a screen saver application should draw when invoked by XCDE.

The *display* argument is the X display. The *window* argument is the address of a pointer to receive the pointer to a list of windows. The *count* argument is the address of an integer to receive the number of elements in the list of windows returned in *window*.

RETURN VALUE

Upon successful completion, the *DtSaverGetWindows*( ) function returns True; otherwise, it returns False.

APPLICATION USAGE

If the *DtSaverGetWindows*( ) function is not successful, failure may be due to a memory allocation error or that the screen saver application was not invoked from XCDE.

The application must use *XtFree*( ) to free data returned in **window*.

SEE ALSO

<Dt/Saver.h>; *XtFree*( ) in the X / Open CAE Specification, **Window Management**: **X Toolkit Intrinsics**.

CHANGE HISTORY

First released in Issue 1.

NAME

DtSessionRestorePath — get a pathname for the application's state information file

SYNOPSIS

```
#include <Dt/Session.h>

Boolean DtSessionRestorePath(Widget widget,
                             char **restorePath,
                             char *restoreFile);
```

DESCRIPTION

The *DtSessionRestorePath*( ) function returns a pathname to an application's state information.

The *widget* argument is the application's top level widget. The *restorePath* argument is the address of the character string to receive the pathname of the application's state information file. The *restoreFile* argument is the filename of the file containing the application state information. This is the filename specified with the **–session** option at application invocation.

RETURN VALUE

Upon successful completion, the *DtSessionRestorePath*( ) function returns True; otherwise, it returns False.

APPLICATION USAGE

The application must support the **–session** command-line option.

The application must use *XtFree*( ) to free data returned in **restorePath*.

SEE ALSO

<Dt/Session.h>, *DtSessionSavePath*( ); *XtFree*( ) in the X / Open CAE Specification, **Window Management: X Toolkit Intrinsics**; X / Open CAE Specification, **Window Management: File Formats and Application Conventions**.

CHANGE HISTORY

First released in Issue 1.

NAME

DtSessionSavePath — get a pathname for saving application state information

SYNOPSIS

```
#include <Dt/Session.h>

Boolean DtSessionSavePath(Widget widget,
                          char **savePath,
                          char **saveFile);
```

DESCRIPTION

The *DtSessionSavePath*( ) function returns the pathname to be used by an application for saving its state information. The information is later used by the application to restore its state.

The *widget* argument is the application's top level widget. The *savePath* argument is the address of the character string to receive the pathname of the state information file to be used by the application for storing its state. The *saveFile* argument is the address of the character string to receive the filename of the file to be used by the application for storing its state.

RETURN VALUE

Upon successful completion, the *DtSessionSavePath*( ) function returns True; otherwise, it returns False.

APPLICATION USAGE

The application should add –**session** *saveFile* when it updates its WM_COMMAND property.

The application must use *XtFree*( ) to free data returned in ∗*savePath* and ∗*saveFile*.

SEE ALSO

<Dt/Session.h>, *DtSessionRestorePath*( ); *XtFree*( ) in the X / Open CAE Specification, **Window Management: X Toolkit Intrinsics**; X / Open CAE Specification, **Window Management: File Formats and Application Conventions**.

CHANGE HISTORY

First released in Issue 1.

3.3 Headers

This section describes the contents of headers used by the XCDE session management service functions, macros and external variables.

Headers contain the definition of symbolic constants, common structures, preprocessor macros and defined types. Each function in Section 3.2 specifies the headers that an application must include in order to use that function. In most cases only one header is required. These headers are present on an application development system; they do not have to be present on the target execution system.

NAME

Dt/Saver.h — screen saver definitions

SYNOPSIS

```
#include <Dt/Saver.h>
```

DESCRIPTION

The **<Dt/Saver.h>** header defines the following as a function:

```
Boolean DtSaverGetWindows(Display *display,
                          Window **window,
                          int *count);
```

CHANGE HISTORY

First released in Issue 1.

NAME

Dt/Session.h — session management services definitions

SYNOPSIS

```
#include <Dt/Session.h>
```

DESCRIPTION

The **<Dt/Session.h>** header defines the following as functions:

```
Boolean DtSessionSavePath(Widget widget,
                          char **save_path,
                          char **save_file);

Boolean DtSessionRestorePath(Widget widget,
                             char **restore_path,
                             char *restore_file);
```

CHANGE HISTORY

First released in Issue 1.

3.4 Actions

This section defines the actions that provide XCDE session management services to support application portability at the C-language source or shell script levels.

NAME

dtsessionaction — XCDE session management actions

SYNOPSIS

```
ExitSession
LockDisplay
ReloadResources
```

DESCRIPTION

The XCDE Session Management Services support the following session management actions:

ExitSession
Exit the user's current session.

LockDisplay
Lock the user's display.

ReloadResources
Reload the user's resources.

These actions can be invoked from an application using the *DtActionInvoke*( ) function or invoked from a command line using the *dtaction* utility.

SEE ALSO

dtaction, *DtActionInvoke*( ) in the **XCDI** specification.

CHANGE HISTORY

First released in Issue 1.

3.5 Capabilities

A conforming implementation of the XCDE session management services supports at least the following capabilities:

1. Conforms to the Required items in the Application Style Checklist in Chapter 20 on page 355.
2. Has been internationalised using the standard interfaces in the X/Open CAE Specification, **System Interfaces and Headers**, **Issue 4, Version 2**, the X/Open CAE Specification, **Window Management**: **Xlib C Language Binding**, and the X/Open CAE Specification, **Motif Toolkit API**, and supports any locale available in the underlying X Window System Xlib implementation.
3. Provides ICCCM 1.1 compliant session management (see the X/Open CAE Specification, **Window Management**: **File Formats and Application Conventions**).
4. Restores an initial *home* or *current* session at session startup. See the Style Management Services Capabilities in Section 13.3 on page 301 for configuration of session startup.
5. Provides session locking on user request. Some systems may provide a user option to perform session locking on X server timeout.
6. Invokes a session screen saver on user request. Some systems may provide a user option to perform screen saving on X server timeout.
7. Saves the *home* session at user request or *current* session at session exit.
8. Displays a confirmation dialog or session selection dialog at session exit. See the Style Management Services Capabilities in Section 13.3 on page 301 for configuration of session exit.

Chapter 4

Help Services

4.1 Introduction

The XCDE help services are a system for developing online help for any XCDE-based application. It allows authors to write, and developers to integrate, online help that includes graphics and text formatting, embedded hyperlinks, and two-way communication with the application.

The services include three main components: a markup language, an application program interface, and an action interface. The *HelpTag* markup language is used to author help topics. It complies with the Standard Generalised Markup Language (SGML), ISO 8879:1986. Various function calls allow applications to use the help services directly. Help actions provide access to help from other components such as the front panel.

4.2 Widgets

This section defines the widget classes that provide XCDE help services to support application portability at the C-language source level.

NAME

DtHelpDialog — DtHelpDialog widget class

SYNOPSIS

```
#include <Dt/HelpDialog.h>
```

DESCRIPTION

The DtHelpDialog widget provides users with functionality for viewing and navigating structured online information (XCDE help volumes). This functionality includes text and graphics rendering, embedded hypertext links and various navigation methods to move through online help information. The widget supports rendering of XCDE help volumes, system manual pages, text files and character string values.

When the user resizes the window, the DtHelpDialog widget dynamically reformats its contents to fit the new window size if the **DtNhelpType** of the contents is DtHELP_TYPE_TOPIC or DtHELP_TYPE_DYNAMIC_STRING. If the **DtNhelpType** of the contents is DtHELP_TYPE_STRING, DtHELP_TYPE_MAN_PAGE or DtHELP_TYPE_FILE, the contents are not reformatted. Instead, scroll bars may appear when the user resizes the window smaller than the help contents. The exact scrollbar behaviour is controlled by the **DtNscrollBarPolicy** resource.

Users can re-specify certain resources for the automatically created widgets and gadgets contained within the DtHelpDialog widget hierarchy. The following list identifies the names of these widgets (or gadgets):

Topic Tree – *TocArea*

Display Area – *DisplayArea*

Button Box – *BtnBox*

The DtHelpDialog widget honours all default and user-specified resource settings, with one exception. In the case where an error occurs due to an invalid request, the DtHelpDialog widget posts the proper error message in its display area and modifies the **DtNhelpType** resource to reflect the current contents of the display area (that is, a string message). Applications and users should set the **DtNhelpType** resource to the appropriate value with each setting of the DtHelpDialog widget.

Classes

The DtHelpDialog widget inherits behaviour and resources from the *Core*, *Composite*, *Constraint*, *XmManager* and *XmBulletinBoard* classes.

The class pointer is **dtHelpDialogClass**.

The class name is *DtHelpDialog*.

New Resources

To reference a resource by name or by class in a **.Xdefaults** file, the application must remove the **DtN** or **DtC** prefix and use the remaining letters. To specify one of the defined values for a resource in a **.Xdefaults** file, the application must remove the **Dt** prefix and use the remaining letters (in either lower case or upper case, but including any underscores between words). The codes in the access column indicate if the given resource can be set at creation time (C), set by using *XtSetValues*( ) (S), retrieved by using *XtGetValues*( ) (G), or is not applicable (N / A).

DtHelpDialog Resource Set				
Name	**Class**	**Type**	**Default**	**Access**
DtNcloseCallback	**DtCCloseCallback**	**XtCallbackList**	NULL	C
DtNcolumns	**DtCColumns**	**Dimension**	70	CSG
DtNhelpFile	**DtCHelpFile**	**char** *	NULL	CSG
DtNhelpOnHelpVolume	**DtCHelpOnHelpVolume**	**char** *	See Definition	C
DtNhelpType	**DtCHelpType**	**unsigned char**	See Definition	CSG
DtNhelpVolume	**DtCHelpVolume**	**char** *	NULL	CSG
DtNhyperLinkCallback	**DtCHyperLinkCallback**	**XtCallbackList**	NULL	C
DtNlocationId	**DtCLocationId**	**char** *	See Definition	CSG
DtNmanPage	**DtCManPage**	**char** *	NULL	CSG
DtNrows	**DtCRows**	**Dimension**	25	CSG
DtNscrollBarPolicy	**DtCScrollBarPolicy**	**unsigned char**	See Definition	C
DtNstringData	**DtCStringData**	**char** *	NULL	CSG
DtNtopicTitle	**DtCTopicTitle**	**char** *	NULL	CSG

DtNcloseCallback

Specifies the list of callbacks called when the user activates the Close button. The callback reason is DtCR_HELP_CLOSE.

DtNcolumns

Specifies the number of columns of text to display in the display area of the DtHelpDialog widget.

DtNhelpFile

Specifies the absolute pathname of a text file to be read and displayed. This resource is used when the **DtNhelpType** is set to DtHELP_TYPE_FILE.

DtNhelpOnHelpVolume

Specifies the help volume that contains the help topics for the help user-interface components in the widget. This is displayed in an instance of the DtHelpDialog widget when the user requests help from within the widget. The default value for this resource is **Help4Help**, which refers to the default-supported help volume. This resource supports absolute pathnames and pathless help volume names. When just a volume name is used, the volume must be placed or linked to one of the default search locations, or one of the two help search path environment variables must be properly set. See the **ENVIRONMENT VARIABLES** section for more information on setting and modifying these variables.

DtNhelpType

Specifies the current topic type. When the value is DtHELP_TYPE_TOPIC, the **DtNlocationId** and **DtNhelpVolume** resources are used and the requested help topic is displayed. When the value is DtHELP_TYPE_STRING or DtHELP_TYPE_-DYNAMIC_STRING, the **DtNstringData** resource is used and the requested string is displayed. When the value is DtHELP_TYPE_FILE, the **DtNhelpFile** resource is used and the requested text file is displayed. When the value is DtHELP_TYPE_-MAN_PAGE, the **DtNmanPage** resource is used and the requested system manual page is displayed. The initial default value is DtHELP_TYPE_TOPIC; however, each time there is a request to display a help topic, text file, manual page or text string, the user should reset **DtNhelpType** to the proper type.

DtNhelpVolume

Specifies the help volume to use. This resource is used in conjunction with the **DtNlocationId** resource to display help topics. This resource supports absolute pathnames and pathless help volume names. When using just a volume name, the volume must be placed in or linked to one of the default search locations, or one of

the two help search path environment variables must be properly set. See the **ENVIRONMENT VARIABLES** section for more information on setting and modifying these variables.

DtNhyperLinkCallback
Specifies the callback that is called when a client-specific hypertext link is activated in the display area of the DtHelpDialog widget. Links are activated when the user presses mouse button 1 over a hypertext link, or presses <return> with the keyboard focus on the hypertext link item. The callback reason is DtCR_HELP_-LINK_ACTIVATE. **DtNhyperLinkCallback** allows applications to register a callback procedure that is used to process one of four hypertext link types: DtHELP_LINK_APP_DEFINE, DtHELP_LINK_TOPIC, DtHELP_LINK_-MAN_PAGE or DtHELP_LINK_TEXT_FILE. For DtHELP_LINK_TOPIC, the callback is made only when the *windowHint* value in the callback structure is DtHELP_NEW_WINDOW.

DtNlocationId
Specifies a help topic to display. Applications reference topics within a help volume using a location ID. Location IDs are author-defined at help volume creation time. Applications use these location IDs to display the desired help topic. The default value for this resource is _HOMETOPIC, which refers to the help volume's top level topic. **DtNhelpVolume** must be set to the help volume in which the corresponding location ID resides, and **DtNhelpType** must be set to DtHELP_TYPE_TOPIC.

DtNmanPage
Specifies the system manual page to display in the current DtHelpDialog widget. This resource is used when the **DtNhelpType** is set to DtHELP_TYPE_MAN_PAGE.

DtNrows
Specifies the number of rows of text to display in the display area of the DtHelpDialog widget.

DtNscrollBarPolicy
Controls the automatic placement of scroll bars in the text display area. If it is set to DtHELP_AS_NEEDED_SCROLLBARS, the scroll bars are displayed only if the display area exceeds the clip area in one or both dimensions. A resource value of DtHELP_STATIC_SCROLLBARS causes the display area to display the scroll bars whenever the DtHelpDialog widget is managed, regardless of the relationship between the clip window and the display area. A value of DtHELP_NO_-SCROLLBARS removes scroll bars from the DtHelpDialog widget. The default value is DtHELP_AS_NEEDED_SCROLLBARS.

DtNstringData
Specifies the string data (**char** *) to display in the current DtHelpDialog widget. This resource is used when the **DtNhelpType** is set to DtHELP_TYPE_STRING.

DtNtopicTitle
Specifies the topic title (**char** *) to be used in conjunction with either the **DtNstringData** or **DtNhelpFile** resource. The topic title is required in order to maintain an accurate and descriptive history list. The topic title is also used as the default heading for the banner page and page header when printing. When printing help topics, this resource may be ignored.

Inherited Resources

The DtHelpDialog widget inherits behaviour and resources from the following named superclasses. For a complete description of each resource, see the entry in X/Open CAE Specification, **Motif Toolkit API** for that superclass.

XmBulletinBoard Resource Set				
Name	**Class**	**Type**	**Default**	**Access**
XmNallowOverlap	**XmCAllowOverlap**	**Boolean**	True	CSG
XmNautoUnmanage	**XmCAutoUnmanage**	**Boolean**	True	CG
XmNbuttonFontList	**XmCButtonFontList**	**XmFontList**	dynamic	CSG
XmNcancelButton	**XmCWidget**	**Widget**	dynamic	SG
XmNdefaultButton	**XmCWidget**	**Widget**	dynamic	SG
XmNdefaultPosition	**XmCDefaultPosition**	**Boolean**	True	CSG
XmNdialogStyle	**XmCDialogStyle**	**unsigned char**	dynamic	CSG
XmNdialogTitle	**XmCDialogTitle**	**XmString**	NULL	CSG
XmNfocusCallback	**XmCCallback**	**XtCallbackList**	NULL	C
XmNlabelFontList	**XmCLabelFontList**	**XmFontList**	dynamic	CSG
XmNmapCallback	**XmCCallback**	**XtCallbackList**	NULL	C
XmNmarginHeight	**XmCMarginHeight**	**Dimension**	10	CSG
XmNmarginWidth	**XmCMarginWidth**	**Dimension**	10	CSG
XmNnoResize	**XmCNoResize**	**Boolean**	False	CSG
XmNresizePolicy	**XmCResizePolicy**	**unsigned char**	XmRESIZE_ANY	CSG
XmNshadowType	**XmCShadowType**	**unsigned char**	XmSHADOW_OUT	CSG
XmNtextFontList	**XmCTextFontList**	**XmFontList**	dynamic	CSG
XmNtextTranslations	**XmCTranslations**	**XtTranslations**	NULL	C
XmNunmapCallback	**XmCCallback**	**XtCallbackList**	NULL	C

XmManager Resource Set				
Name	**Class**	**Type**	**Default**	**Access**
XmNbottom-ShadowColor	**XmCBottom-ShadowColor**	**Pixel**	dynamic	CSG
XmNbottom-ShadowPixmap	**XmCBottom-ShadowPixmap**	**Pixmap**	XmUNSPECIFIED-_PIXMAP	CSG
XmNforeground	**XmCForeground**	**Pixel**	dynamic	CSG
XmNhelpCallback	**XmCCallback**	**XtCallbackList**	NULL	C
XmNhighlightColor	**XmCHighlightColor**	**Pixel**	dynamic	CSG
XmNhighlightPixmap	**XmCHighlightPixmap**	**Pixmap**	dynamic	CSG
XmNinitialFocus	**XmCInitialFocus**	**Widget**	dynamic	CSG
XmNnavigationType	**XmCNavigationType**	**XmNavigation-Type**	XmTAB_GROUP	CSG
XmNshadowThickness	**XmCShadowThickness**	**Dimension**	dynamic	CSG
XmNstringDirection	**XmCStringDirection**	**XmString-Direction**	dynamic	CG
XmNtopShadowColor	**XmCTopShadowColor**	**Pixel**	dynamic	CSG
XmNtopShadowPixmap	**XmCTopShadowPixmap**	**Pixmap**	dynamic	CSG
XmNtraversalOn	**XmCTraversalOn**	**Boolean**	True	CSG
XmNunitType	**XmCUnitType**	**unsigned char**	dynamic	CSG
XmNuserData	**XmCUserData**	**XtPointer**	NULL	CSG

Composite Resource Set				
Name	**Class**	**Type**	**Default**	**Access**
XmNchildren	**XmCReadOnly**	**WidgetList**	NULL	G
XmNinsertPosition	**XmCInsertPosition**	**XtOrderProc**	NULL	CSG
XmNnumChildren	**XmCReadOnly**	**Cardinal**	0	G

Core Resource Set				
Name	**Class**	**Type**	**Default**	**Access**
XmNaccelerators	**XmCAccelerators**	**XtAccelerators**	dynamic	N/A
XmNancestorSensitive	**XmCSensitive**	**Boolean**	dynamic	G
XmNbackground	**XmCBackground**	**Pixel**	dynamic	CSG
XmNbackgroundPixmap	**XmCPixmap**	**Pixmap**	XmUNSPECIFIED-_PIXMAP	CSG
XmNborderColor	**XmCBorderColor**	**Pixel**	XtDefaultForeground	CSG
XmNborderPixmap	**XmCPixmap**	**Pixmap**	XmUNSPECIFIED-_PIXMAP	CSG
XmNborderWidth	**XmCBorderWidth**	**Dimension**	0	CSG
XmNcolormap	**XmCColormap**	**Colormap**	dynamic	CG
XmNdepth	**XmCDepth**	**int**	dynamic	CG
XmNdestroyCallback	**XmCCallback**	**XtCallbackList**	NULL	C
XmNheight	**XmCHeight**	**Dimension**	dynamic	CSG
XmNinitialResources-Persistent	**XmCInitialResources-Persistent**	**Boolean**	True	C
XmNmappedWhen-Managed	**XmCMappedWhen-Managed**	**Boolean**	True	CSG
XmNscreen	**XmCScreen**	**Screen** *	dynamic	CG
XmNsensitive	**XmCSensitive**	**Boolean**	True	CSG
XmNtranslations	**XmCTranslations**	**XtTranslations**	dynamic	CSG
XmNwidth	**XmCWidth**	**Dimension**	dynamic	CSG
XmNx	**XmCPosition**	**Position**	0	CSG
XmNy	**XmCPosition**	**Position**	0	CSG

Callback Information

A pointer to the following structure is passed to each callback:

```
typedef struct {
        int    reason;
        XEvent *event;
        char   *locationId;
        char   *helpVolume;
        char   *specification;
        int    hyperType;
        int    windowHint;
} DtHelpDialogCallbackStruct;
```

The *reason* argument indicates why the callback was invoked.

The *event* argument points to the **XEvent** that triggered the callback.

The *locationId* argument indicates the **DtNlocationId** for the current topic. This value is NULL whenever the *hyperType* value is not DtHELP_LINK_TOPIC or DtHELP_LINK_APP_DEFINE.

The *helpVolume* argument indicates the current help volume. This value is NULL whenever the *hyperType* value is not DtHELP_LINK_TOPIC or DtHELP_LINK_APP_DEFINE.

The *specification* argument indicates any author-defined data that was contained within the hypertext link selected. This value returns NULL if no author-defined data was given. For hyperlinks of type DtHELP_MAN_PAGE, the *specification* argument contains the name of the manual page. For hyperlinks of type DtHELP_LINK_TEXT_FILE, the *specification* argument contains that name of the file.

The *hyperType* argument indicates the hypertext link type. Possible values are: DtHELP_LINK_-TOPIC, DtHELP_LINK_MAN_PAGE, DtHELP_LINK_APP_DEFINE or DtHELP_LINK_-TEXT_FILE.

The *windowHint* argument indicates a hint for the type of window (quick help, existing or new window) to use. This value contains one of the following three types: DtHELP_POPUP_WINDOW, DtHELP_CURRENT_WINDOW or DtHELP_NEW_WINDOW.

Additional Behaviour

The DtHelpDialog widget has the additional behaviour described below:

<*MAny*> <*KCancel*>
: Calls the active callbacks for the Close button. If a <*BDrag*> for either a selection or scrollbar movement is in process, the <*KCancel*> aborts that action.

<*KSpace*>, <*KActivate*> or <*BSelect*> in Topic Tree Text
: Opens the help topic currently selected, displays that topic in the display area and updates the topic tree to match the newly displayed topic.

<*KSpace*>, <*KActivate*> or <*BSelect*> in Display Area Text
: Invokes the hypertext link that contains the current selection.

<*DoubleClick*> in Topic Tree or Display Area Text
: Ignored.

<*BDrag*> in Topic Tree or Display Area Text
: Selects the text from the drag start point to the drag end point. Moving and holding the <*BDrag*> outside the topic tree or display area scrolls the window, selecting the newly exposed text.

<*MCtrl*> or <*MShift*> <*BSelect*> in Topic Tree Text

<*MCtrl*> <*KSpace*> in Topic Tree Text

<*MCtrl*> <*KActivate*> in Topic Tree Text
: Invokes the **DtNhyperLinkCallback** for the DtHelpDialog widget, setting the *helpType* to DtHELP_LINK_TOPIC, the *windowHint* to DtHELP_NEW_WINDOW, the *helpVolume* to the current volume name and the *locationId* to the selected item's location ID. If no **DtNhyperLinkCallback** is supplied, the action is ignored.

<*MCtrl*> or <*MShift*> <*BSelect*> in Display Area Hypertext Link Text
: Invokes the **DtNhyperLinkCallback** for the DtHelpDialog widget, honouring all existing link settings, but forces the *windowHint* to DtHELP_NEW_WINDOW. If no **DtNhyperLinkCallback** is supplied, the hypertext link is handled internally.

<*KSelectAll*> in Display Area or Topic Tree
: Selects all text within the topic tree area or display area.

<*KDeSelectAll*> in Display Area or Topic Tree
: Deselects all text within the topic tree area or display area.

<*KCopy*> in the Display Area or Topic Tree
: Copies the currently selected text to the clipboard.

<*KPageDown*> or <*MCtrl*> <*KDown*> in Display Area or Topic Tree
: Displays the next page of text.

<*KPageLeft*> or <*MCtrl*> <*KLeft*> in Display Area or Topic Tree
: Scrolls the information to the left.

<*KPageRight*> or <*MCtrl*> <*KRight*> in Display Area or Topic Tree
Scrolls the information to the right.

<*KPageUp*> or <*MCtrl*> <*KUp*> in Display Area or Topic Tree
Displays the previous page of information.

<*KBeginData*> in the Display Area or Topic Tree
Displays the first page of information.

<*KEndData*> in the Display Area or Topic Tree
Displays the last page of information.

The following operations are supported, but the key bindings are implementation-dependent:

<*implementation-dependent*>
Moves the traversal highlight up, down, left or right to the next hypertext link item.

Virtual Bindings

The bindings for virtual keys are implementation-dependent.

ENVIRONMENT VARIABLES

The DtHelpDialog widget uses two environment variables for locating help volumes within the desktop environment:

DTHELPSEARCHPATH
The system search path environment variable for locating help volumes on local and remote mounted systems.

DTHELPUSERSEARCHPATH
The search path environment variable for locating user-specific help volumes on local and remote mounted systems.

The environment variables contain colon-separated lists of directory paths. Each directory path can contain both environment variable names as well as special field descriptors that are expanded at runtime.

Field descriptors consist of a percent-sign character (%) followed by a single character. Field descriptors and their substitution values are:

%H Replaced with the current volume name being searched for.

%L Replaced with the current value of the *LANG* environment variable.

%% Replaced with a single %.

The default value for *DTHELPUSERSEARCHPATH* is:

```
$HOME/.dt/help/$DTUSERSESSION/%H:
$HOME/.dt/help/$DTUSERSESSION/%H.sdl:
$HOME/.dt/help/%H:
$HOME/.dt/help/%H.sdl:
```

The *DTHELPUSERSEARCHPATH* is first searched for the requested volume. If the volume is not found, the *DTHELPSEARCHPATH* value is searched.

The default value for *DTHELPSEARCHPATH* path is:

```
/etc/dt/appconfig/help/%L/%H:
/etc/dt/appconfig/help/%L/%H.sdl:
/etc/dt/appconfig/help/C/%H:
/etc/dt/appconfig/help/C/%H.sdl:
/usr/dt/appconfig/help/%L/%H:
/usr/dt/appconfig/help/%L/%H.sdl:
/usr/dt/appconfig/help/C/%H:
/usr/dt/appconfig/help/C/%H.sdl:
```

SEE ALSO

<Dt/HelpQuickD.h>, **<Dt/Help.h>**, *DtCreateHelpQuickDialog*( ), *DtHelpSetCatalogName*( ); *XmManager* and *XmBulletinBoard* in the X/Open CAE Specification, **Motif Toolkit API**; Section 4.6 on page 65.

CHANGE HISTORY

First released in Issue 1.

NAME

DtHelpQuickDialog — DtHelpQuickDialog widget class

SYNOPSIS

```
#include <Dt/HelpQuickD.h>
```

DESCRIPTION

The DtHelpQuickDialog widget provides users with a constrained set of functionality for viewing and and navigating structured online information (XCDE help volumes). This functionality includes text and graphics rendering, embedded hypertext links and limited navigation methods to move through online help information. The widget supports rendering of XCDE help volume, system manual pages, text files and character string values.

When the user resizes the window, the DtHelpQuickDialog widget dynamically reformats its contents to fit the new window size if the **DtNhelpType** of the contents is DtHELP_TYPE_TOPIC or DtHELP_TYPE_DYNAMIC_STRING. If the **DtNhelpType** of the contents is DtHELP_TYPE_STRING, DtHELP_TYPE_MAN_PAGE or DtHELP_TYPE_FILE, the contents are not reformatted. Instead, scroll bars may appear when the user resizes the window smaller than the help contents. The exact scrollbar behaviour is controlled by the **DtNscrollBarPolicy** resource.

Users can re-specify certain resources for the automatically created widget contained within the DtHelpQuickDialog hierarchy. The following is the name of the widget:

Display Area – *DisplayArea*

The DtHelpQuickDialog widget honours all default and user-specified resource settings, with one exception. In the case where an error occurs due to a non-valid request, the widget posts the proper error message in its display area, and modifies the **DtNhelpType** to reflect the current contents of the display area (that is, a string message). Applications and users should set the **DtNhelpType** to the appropriate value with each setting of the help value.

Classes

The DtHelpQuickDialog widget inherits behaviour and resources from the *Core*, *Composite*, *Constraint*, *XmManager* and *XmBulletinBoard* classes.

The class pointer is **dtHelpQuickDialogClass**.

The class name is *DtHelpQuickDialog*.

New Resources

To reference a resource by name or by class in a **.Xdefaults** file, the application must remove the **DtN** or **DtC** prefix and use the remaining letters. To specify one of the defined values for a resource in a **.Xdefaults** file, the application must remove the **Dt** prefix and use the remaining letters (in either lower case or upper case, but including any underscores between words). The codes in the access column indicate if the given resource can be set at creation time (C), set by using *XtSetValues*( ) (S), retrieved by using *XtGetValues*( ) (G), or is not applicable (N / A).

DtHelpQuickDialog Resource Set				
Name	**Class**	**Type**	**Default**	**Access**
DtNbackLabelString	**DtCBackLabelString**	**XmString**	Back Track	CSG
DtNcloseCallback	**DtCCloseBtnCallback**	**XtCallbackList**	NULL	C
DtNcloseLabelString	**DtCcloseLabelString**	**XmString**	Close	CSG
DtNcolumns	**DtCColumns**	**Dimension**	50	CSG
DtNhelpFile	**DtCHelpFile**	**char** *	NULL	CSG
DtNhelpLabelString	**DtCHelpLabelString**	**XmString**	Help	CSG
DtNhelpOnHelp-Volume	**DtCHelpOnHelp-Volume**	**char** *	See Definition	C
DtNhelpType	**DtCHelpType**	**unsigned char**	DtHELP_TYPE-_TOPIC	CSG
DtNhelpVolume	**DtCHelpVolume**	**char** *	NULL	CSG
DtNhyperLink-Callback	**DtCHyperLink-Callback**	**XtCallbackList**	NULL	C
DtNlocationId	**DtCLocationId**	**char** *	See Definition	CSG
DtNmanPage	**DtCManPage**	**char** *	NULL	CSG
DtNminimizeButtons	**DtCMinimizeButtons**	**Boolean**	True	C
DtNmoreLabelString	**DtCMoreLabelString**	**XmString**	More	CSG
DtNprintLabelString	**DtCPrintLabelString**	**XmString**	Print...	CSG
DtNrows	**DtCRows**	**Dimension**	15	CSG
DtNscrollBarPolicy	**DtCScrollBarPolicy**	**unsigned char**	DtHELP_AS-_NEEDED-_SCROLLBARS	C
DtNstringData	**DtCStringData**	**char** *	NULL	CSG
DtNtopicTitle	**DtCTopicTitle**	**char** *	NULL	CSG

DtNbackLabelString
Specifies the string label for the Back button.

DtNcloseCallback
Specifies the list of callbacks called when the application activates the Close button. The callback reason is DtCR_HELP_CLOSE.

DtNcloseLabelString
Specifies the string label for the Close button.

DtNcolumns
Specifies the number of columns of text to display in the DtHelpQuickDialog widget display area.

DtNhelpFile
Specifies the absolute pathname of a text file to be read and displayed. This resource is used when the **DtNhelpType** is set to DtHELP_TYPE_FILE. The topic title is required in order to maintain an accurate and descriptive history list.

DtNhelpLabelString
Specifies the string label for the Help button.

DtNhelpOnHelpVolume
Specifies the help volume that contains the help topics for the help user interface components in the widget. This is displayed in an instance of the DtHelpQuickDialog widget when the user requests help from within the widget. The default value for this resource is **Help4Help**, which refers to the default supported help volume. This resource supports absolute pathnames and pathless help volume names. When just a volume name is used, the volume must be placed

or linked to one of the default search locations, or one of the two help search path environment variables must be properly set. See the **ENVIRONMENT VARIABLES** section for more information on setting and modifying these variables.

DtNhelpType
Specifies the current topic type. When the value is DtHELP_TYPE_TOPIC, the **DtNlocationId** and **DtNhelpVolume** resources are used and the requested help topic is displayed. When the value is DtHELP_TYPE_STRING or DtHELP_TYPE_-DYNAMIC_STRING, the **DtNstringData** resource is used and the requested string is displayed. When the value is DtHELP_TYPE_FILE, the **DtNhelpFile** resource is used and the requested text file is displayed. When the value is DtHELP_TYPE_-MAN, the **DtNmanPage** resource is used and the requested system manual page is displayed. The initial default value is DtHELP_TYPE_TOPIC; however, each time there is a request to display a help topic, text file, manual page or text string, the user should reset **DtNhelpType** to the proper type.

DtNhelpVolume
Specifies the help volume to use. This resource is used in conjunction with the **DtNlocationId** resource to display help topics. This resource supports absolute pathnames and pathless help volume names. When using just a volume name, the volume must be placed in or linked to one of the default search locations, or one of the two help search path environment variables must be properly set. See the **ENVIRONMENT VARIABLES** section for more information on setting and modifying these variables.

DtNhyperLinkCallback
Specifies the callback called when a client-specific hypertext link type is activated in the display area of the DtHelpQuickDialog widget. Links are activated when the user presses mouse button 1 over a hypertext link, or presses <KActivate> with the keyboard focus on the hypertext link item. The callback reason is DtCR_HELP_LINK_ACTIVATE. **DtNhyperLinkCallback** allows applications to register a callback procedure used to process one of four hypertext link types: DtHELP_LINK_APP_DEFINE, DtHELP_LINK_TOPIC, DtHELP_LINK_-MAN_PAGE or DtHELP_LINK_TEXT_FILE. For the DtHELP_LINK_TOPIC, the callback is made only when the *windowHint* value in the callback structure is DtHELP_NEW_WINDOW.

DtNlocationId
Specifies a help topic to display. Applications reference topics within a help volume using a location ID. Location IDs are author-defined at help volume creation time. Applications use these location IDs to display the desired help topic. The default value for this resource is _HOMETOPIC, which refers to the help volume's top level topic. **DtNhelpVolume** must be set to the help volume in which the corresponding location ID resides, and **DtNhelpType** must be set to DtHELP_TYPE_TOPIC.

DtNmanPage
Specifies the system manual page to display in the current DtHelpQuickDialog widget. This resource is used when the **DtNhelpType** is set to DtHELP_TYPE_-MAN_PAGE.

DtNminimizeButtons
Sets the buttons to the width of the widest button and the height of the tallest button if False. If True, button width and height are not modified.

DtNmoreLabelString
Specifies the string label for the More button.

DtNprintLabelString
Specifies the string label for the Print button.

DtNrows Specifies the number of rows of text to display in the display area of the DtHelpQuickDialog widget.

DtNscrollBarPolicy
Controls the automatic placement of scroll bars in the text display area. If it is set to DtHELP_AS_NEEDED_SCROLLBARS, the scroll bars are displayed only if the display area exceeds the clip area in one or both dimensions. A resource value of DtHELP_STATIC_SCROLLBARS causes the display area to display the scroll bars whenever the DtHelpQuickDialog widget is managed, regardless of the relationship between the clip window and the display area. A value of DtHELP_NO_-SCROLLBARS removes scroll bars from the DtHelpQuickDialog widget. The default value is DtHELP_AS_NEEDED_SCROLLBARS.

DtNstringData
Specifies the string data (**char** *) to display in the current DtHelpQuickDialog widget. This resource is used when the **DtNhelpType** is set to DtHELP_TYPE_-STRING.

DtNtopicTitle
Specifies the topic title (**char** *) to be used in conjunction with either the **DtNstringData** or **DtNhelpFile** resource. The topic title is required in order to maintain an accurate and descriptive history list. The topic title is also used as the default heading for the banner page and page header when printing. When printing help topics, this resource may be ignored.

Inherited Resources

The DtHelpQuickDialog widget inherits behaviour and resources from the following named superclasses. For a complete description of each resource, see the entry in X/Open CAE Specification, **Motif Toolkit API** for that superclass.

XmBulletinBoard Resource Set				
Name	**Class**	**Type**	**Default**	**Access**
XmNallowOverlap	**XmCAllowOverlap**	**Boolean**	True	CSG
XmNautoUnmanage	**XmCAutoUnmanage**	**Boolean**	True	CG
XmNbuttonFontList	**XmCButtonFontList**	**XmFontList**	dynamic	CSG
XmNcancelButton	**XmCWidget**	**Widget**	dynamic	SG
XmNdefaultButton	**XmCWidget**	**Widget**	dynamic	SG
XmNdefaultPosition	**XmCDefaultPosition**	**Boolean**	True	CSG
XmNdialogStyle	**XmCDialogStyle**	**unsigned char**	dynamic	CSG
XmNdialogTitle	**XmCDialogTitle**	**XmString**	NULL	CSG
XmNfocusCallback	**XmCCallback**	**XtCallbackList**	NULL	C
XmNlabelFontList	**XmCLabelFontList**	**XmFontList**	dynamic	CSG
XmNmapCallback	**XmCCallback**	**XtCallbackList**	NULL	C
XmNmarginHeight	**XmCMarginHeight**	**Dimension**	10	CSG
XmNmarginWidth	**XmCMarginWidth**	**Dimension**	10	CSG
XmNnoResize	**XmCNoResize**	**Boolean**	False	CSG
XmNresizePolicy	**XmCResizePolicy**	**unsigned char**	XmRESIZE_ANY	CSG
XmNshadowType	**XmCShadowType**	**unsigned char**	XmSHADOW_OUT	CSG
XmNtextFontList	**XmCTextFontList**	**XmFontList**	dynamic	CSG
XmNtextTranslations	**XmCTranslations**	**XtTranslations**	NULL	C
XmNunmapCallback	**XmCCallback**	**XtCallbackList**	NULL	C

XmManager Resource Set				
Name	**Class**	**Type**	**Default**	**Access**
XmNbottom-ShadowColor	**XmCBottom-ShadowColor**	**Pixel**	dynamic	CSG
XmNbottom-ShadowPixmap	**XmCBottom-ShadowPixmap**	**Pixmap**	XmUNSPECIFIED-_PIXMAP	CSG
XmNforeground	**XmCForeground**	**Pixel**	dynamic	CSG
XmNhelpCallback	**XmCCallback**	**XtCallbackList**	NULL	C
XmNhighlightColor	**XmCHighlightColor**	**Pixel**	dynamic	CSG
XmNhighlightPixmap	**XmCHighlightPixmap**	**Pixmap**	dynamic	CSG
XmNinitialFocus	**XmCInitialFocus**	**Widget**	dynamic	CSG
XmNnavigationType	**XmCNavigationType**	**XmNavigation-Type**	XmTAB_GROUP	CSG
XmNshadowThickness	**XmCShadowThickness**	**Dimension**	dynamic	CSG
XmNstringDirection	**XmCStringDirection**	**XmString-Direction**	dynamic	CG
XmNtopShadowColor	**XmCTopShadowColor**	**Pixel**	dynamic	CSG
XmNtopShadowPixmap	**XmCTopShadowPixmap**	**Pixmap**	dynamic	CSG
XmNtraversalOn	**XmCTraversalOn**	**Boolean**	True	CSG
XmNunitType	**XmCUnitType**	**unsigned char**	dynamic	CSG
XmNuserData	**XmCUserData**	**XtPointer**	NULL	CSG

Composite Resource Set				
Name	**Class**	**Type**	**Default**	**Access**
XmNchildren	**XmCReadOnly**	**WidgetList**	NULL	G
XmNinsertPosition	**XmCInsertPosition**	**XtOrderProc**	NULL	CSG
XmNnumChildren	**XmCReadOnly**	**Cardinal**	0	G

Core Resource Set				
Name	**Class**	**Type**	**Default**	**Access**
XmNaccelerators	**XmCAccelerators**	**XtAccelerators**	dynamic	N/A
XmNancestorSensitive	**XmCSensitive**	**Boolean**	dynamic	G
XmNbackground	**XmCBackground**	**Pixel**	dynamic	CSG
XmNbackgroundPixmap	**XmCPixmap**	**Pixmap**	XmUNSPECIFIED-_PIXMAP	CSG
XmNborderColor	**XmCBorderColor**	**Pixel**	XtDefaultForeground	CSG
XmNborderPixmap	**XmCPixmap**	**Pixmap**	XmUNSPECIFIED-_PIXMAP	CSG
XmNborderWidth	**XmCBorderWidth**	**Dimension**	0	CSG
XmNcolormap	**XmCColormap**	**Colormap**	dynamic	CG
XmNdepth	**XmCDepth**	**int**	dynamic	CG
XmNdestroyCallback	**XmCCallback**	**XtCallbackList**	NULL	C
XmNheight	**XmCHeight**	**Dimension**	dynamic	CSG
XmNinitialResources-Persistent	**XmCInitialResources-Persistent**	**Boolean**	True	C
XmNmappedWhen-Managed	**XmCMappedWhen-Managed**	**Boolean**	True	CSG
XmNscreen	**XmCScreen**	**Screen** *	dynamic	CG
XmNsensitive	**XmCSensitive**	**Boolean**	True	CSG
XmNtranslations	**XmCTranslations**	**XtTranslations**	dynamic	CSG
XmNwidth	**XmCWidth**	**Dimension**	dynamic	CSG
XmNx	**XmCPosition**	**Position**	0	CSG
XmNy	**XmCPosition**	**Position**	0	CSG

Callback Information

A pointer to the following structure is passed to each callback:

```
typedef struct {
        int    reason;
        XEvent *event;
        char   *locationId;
        char   *helpVolume;
        char   *specification;
        int    hyperType;
        int    windowHint;
} DtHelpDialogCallbackStruct;
```

The *reason* argument indicates why the callback was invoked.

The *event* argument points to the **XEvent** that triggered the callback.

The *locationId* argument indicates the **DtNlocationId** for the current topic. This value is NULL whenever the *hyperType* value is not DtHELP_LINK_TOPIC or DtHELP_LINK_APP_DEFINE.

The *helpVolume* argument indicates the current help volume. This value returns NULL whenever the *hyperType* value is not DtHELP_LINK_TOPIC or DtHELP_LINK_APP_DEFINE.

The *specification* argument indicates any author-defined data contained within the selected hypertext link. This value returns NULL if no author-defined data was given. For hyperlinks of type DtHELP_MAN_PAGE, the *specification* argument contains the name of the manual page. For hyperlinks of type DtHELP_LINK_TEXT_FILE, the *specification* argument contains that name of the file.

The *hyperType* argument indicates the hypertext link type. Possible values are: DtHELP_LINK_-TOPIC, DtHELP_LINK_MAN_PAGE, DtHELP_LINK_APP_DEFINE or DtHELP_LINK_-TEXT_FILE.

The *windowHint* argument indicates a hint for the type of window (current window, DtHelpQuickDialog widget window or new window) to use. This value contains one of the following three types: DtHELP_POPUP_WINDOW, DtHELP_CURRENT_WINDOW or DtHELP_NEW_WINDOW.

Additional Behaviour

The DtHelpQuickDialog widget has the additional behaviour described below:

<*MAny*> <*KCancel*>
: Calls the active callbacks for the Close button. If a <*BDrag*> for either a selection or scrollbar movement is in process, the *KCancel* aborts that action.

<*KSpace*>, <*KActivate*> or <*BSelect*> in Display Area Text
: Invokes the hypertext link that contains the current selection.

<*DoubleClick*> in Display Area Text
: Ignored.

<*BDrag*> in Display Area Text
: Selects the text from the drag start point to the drag end point. Moving and holding the <*BDrag*> outside the topic tree or display area, scrolls the window, selecting the newly exposed text.

<*Close Button Activated*>
: Closes the DtHelpQuickDialog widget, and calls the **DtNcloseCallback** callbacks with reason DtCR_HELP_CLOSE.

<*Backtrack Button Activated*>
: Forces the DtHelpQuickDialog widget to display the data previously displayed in the dialog. If the current item was the first item displayed in the DtHelpQuickDialog widget, the Backtrack button is insensitive.

<*Print Button Activated*>
: Forces the DtHelpQuickDialog widget to display the Help-Print dialog.

<*Help Button Activated*>
: Forces the DtHelpQuickDialog widget to display the Help-On-Help dialog.

<*MCtrl*> or <*MShift*> <*BSelect*> in Display Area Hypertext Link Text
<*MCtrl*> <*KSpace*> in Display Area Hypertext Link Text
<*MCtrl*> <*KActivate*> in Display Area Hypertext Link Text
: Invokes the **DtNhyperLinkCallback** for the DtHelpQuickDialog widget, honouring all existing link settings, but forces the *windowHint* to DtHELP_NEW_WINDOW. If no **DtNhyperLinkCallback** was supplied, the hypertext link is handled internally.

<*KSelectAll*> in Display Area
: Selects all text within the display area.

<*KDeSelectAll*> in Display Area
: Deselects all text within the display area.

<*KCopy*> in the Display Area
: Copies the currently selected text to the clipboard.

<*KPageDown*> or <*MCtrl*> <*KDown*> in the Display Area
Displays the next page of text.

<*KPageLeft*> or <*MCtrl*> <*KLeft*> in Display Area
Scrolls the information to the left.

<*KPageRight*> or <*MCtrl*> <*KRight*> in Display Area
Scrolls the information to the right.

<*KPageUp*> or <*MCtrl*> <*KUp*> in Display Area
Displays the previous page of information.

<*KBeginData*> in the Display Area
Displays the first page of information.

<*KEndData*> in the Display Area
Displays the last page of information.

The following operations are supported, but the key bindings are implementation-dependent:

<*implementation-dependent*>
Moves the traversal highlight up, down, left or right to the next hypertext link item.

Virtual Bindings

The bindings for virtual keys are implementation-dependent.

ENVIRONMENT VARIABLES

The DtHelpQuickDialog widget uses two environment variables for locating help volumes within the desktop environment:

DTHELPSEARCHPATH
The system search path environment variable for locating help volumes on local and remote mounted systems.

DTHELPUSERSEARCHPATH
The search path environment variable for locating user-specific help volumes on local and remote mounted systems.

The environment variables contain colon-separated lists of directory paths. Each directory path can contain both environment variable names as well as special field descriptors that are expanded at runtime.

Field descriptors consist of a percent-sign character (%) followed by a single character. Field descriptors and their substitution values are:

%H Replaced with the current volume name being searched for.

%L Replaced with the current value of the *LANG* environment variable.

%% Replaced with a single %.

The default value for *DTHELPUSERSEARCHPATH* is:

```
$HOME/.dt/help/$DTUSERSESSION/%H:
$HOME/.dt/help/$DTUSERSESSION/%H.sdl:
$HOME/.dt/help/%H:
$HOME/.dt/help/%H.sdl:
```

The *DTHELPUSERSEARCHPATH* is first searched for the requested volume. If the volume is not found, the *DTHELPSEARCHPATH* value is searched.

The default value for *DTHELPSEARCHPATH* path is:

```
/etc/dt/appconfig/help/%L/%H:
/etc/dt/appconfig/help/%L/%H.sdl:
/etc/dt/appconfig/help/C/%H:
/etc/dt/appconfig/help/C/%H.sdl:
/usr/dt/appconfig/help/%L/%H:
/usr/dt/appconfig/help/%L/%H.sdl:
/usr/dt/appconfig/help/C/%H:
/usr/dt/appconfig/help/C/%H.sdl:
```

SEE ALSO

<Dt/HelpDialog.h>, **<Dt/Help.h>**, *DtCreateHelpDialog*( ), *DtHelpSetCatalogName*( ), *DtHelpQuickDialogGetChild*( ); *XmManager* and *XmBulletinBoard* in the X / Open CAE Specification, **Motif Toolkit API**; Section 4.6 on page 65.

CHANGE HISTORY

First released in Issue 1.

4.3 Functions

This section defines the functions, macros and external variables that provide XCDE help services to support application portability at the C-language source level.

NAME

DtCreateHelpDialog — create a general DtHelpDialog widget

SYNOPSIS

```
#include <Dt/HelpDialog.h>

Widget DtCreateHelpDialog(Widget parent,
                          String name,
                          ArgList arglist,
                          Cardinal argcount);
```

DESCRIPTION

The *DtCreateHelpDialog*( ) function is a convenience function that creates a DtHelpDialog widget.

The *parent* argument specifies the parent widget ID.

The *name* argument specifies the name of the created widget.

The *arglist* argument specifies the argument list.

The *argcount* argument specifies the number of attribute and value pairs in the argument list (*arglist*).

RETURN VALUE

Upon successful completion, the *DtCreateHelpDialog*( ) function returns an XmBulletinBoard widget whose parent is a dialog shell widget; otherwise, it returns an undefined widget value. The dialog shell is the DtHelpDialog widget's top level.

SEE ALSO

<Dt/HelpDialog.h>, **<Dt/Help.h>**, *DtCreateHelpQuickDialog*( ), *DtHelpSetCatalogName*( ), *XmBulletinBoard* in the X/Open CAE Specification, **Motif Toolkit API**.

CHANGE HISTORY

First released in Issue 1.

NAME

DtCreateHelpQuickDialog — create a DtHelpQuickDialog widget

SYNOPSIS

```
#include <Dt/HelpQuickD.h>

Widget DtCreateHelpQuickDialog(Widget parent,
                               String name,
                               ArgList arglist,
                               Cardinal argcount);
```

DESCRIPTION

The *DtCreateHelpQuickDialog*( ) function is a convenience function that creates a DtHelpQuickDialog widget.

The *parent* argument specifies the parent widget ID.

The *name* argument specifies the name of the created widget.

The *arglist* argument specifies the argument list.

The *argcount* argument specifies the number of attribute and value pairs in the argument list (*arglist*).

RETURN VALUE

Upon successful completion, the *DtCreateHelpQuickDialog*( ) function returns an XmBulletinBoard widget whose parent is a dialog shell widget; otherwise, it returns an undefined widget value. The dialog shell is the DtHelpQuickDialog widget's top level.

SEE ALSO

<Dt/HelpDialog.h>, **<Dt/Help.h>**, *DtCreateHelpDialog*( ), *DtHelpSetCatalogName*( ), *DtHelpQuickDialogGetChild*( ), *XmBulletinBoard* in the X / Open CAE Specification, **Motif Toolkit API**.

CHANGE HISTORY

First released in Issue 1.

NAME

DtHelpQuickDialogGetChild — get child of DtHelpQuickDialog widget

SYNOPSIS

```
#include <Dt/HelpQuickD.h>

Widget DtHelpQuickDialogGetChild(Widget widget,
                                 int child);
```

DESCRIPTION

The *DtHelpQuickDialogGetChild*( ) function accesses a component within a DtHelpQuickDialog widget.

The *widget* argument specifies the DtHelpQuickDialog widget instance.

The *child* argument specifies which DtHelpQuickDialog widget child the widget ID is for. The following are valid values for the *child* argument:

```
DtHELP_QUICK_CLOSE_BUTTON
DtHELP_QUICK_PRINT_BUTTON
DtHELP_QUICK_HELP_BUTTON
DtHELP_QUICK_SEPARATOR
DtHELP_QUICK_MORE_BUTTON
DtHELP_QUICK_BACK_BUTTON
```

RETURN VALUE

Upon successful completion, the *DtHelpQuickDialogGetChild*( ) function returns the widget ID of the requested child of the DtHelpQuickDialog widget; otherwise, it returns NULL if either the function call fails, or an invalid value was passed in for the *child* argument.

SEE ALSO

<Dt/HelpQuickD.h>, *DtCreateHelpQuickDialog*( ); *XmWarning* in the X/Open CAE Specification, **Motif Toolkit API**.

CHANGE HISTORY

First released in Issue 1.

NAME

DtHelpReturnSelectedWidgetId — select a widget or gadget

SYNOPSIS

```
#include <Dt/Help.h>

int DtHelpReturnSelectedWidgetId(Widget parent,
                                 String cursor,
                                 Widget *widget);
```

DESCRIPTION

The *DtHelpReturnSelectedWidgetId*( ) function provides an interface for users to select a component within an application.

This function grabs the pointer and returns the widget within which a button press occurs. Pressing the escape key (ESC) aborts this function.

The *parent* argument specifies the widget ID to use as the basis of the interaction, usually a top level shell.

The *cursor* argument specifies the cursor to be used for the pointer during the interaction. If a NULL value is used, *DtHelpReturnSelectedWidgetId*( ) uses a default cursor value.

The *widget* argument is the return value (for example, the selected widget). A NULL value is returned on error.

The *DtHelpReturnSelectedWidgetId*( ) function allows applications to get the widget ID for any widget in their user interface that the user has selected via the pointer. The application can then directly display a help topic based on the selected widget, or dynamically construct some help information based on the current context of the selected item. At any point while the question mark cursor is displayed, the user can select the escape key to abort the function call, and a NULL value is returned. If the user selects any item outside the current applications windows, an error status is returned along with a NULL value for the *widget* argument.

RETURN VALUE

Upon successful completion, the *DtHelpReturnSelectedWidgetId*( ) function returns one of the following status values:

DtHELP_SELECT_ERROR
: An error occurred while attempting to process the function.

DtHELP_SELECT_INVALID
: The user selected an invalid component that is not contained in the current widget hierarchy.

DtHELP_SELECT_ABORT
: The user aborted the function (for example, pressed the escape key), and a NULL widget value is passed back.

DtHELP_SELECT_VALID
: The user selected a valid component within the application, and the *widget* argument is the ID of the selected component.

SEE ALSO

<Dt/Help.h>, *DtCreateHelpQuickDialog*( ), *DtCreateHelpDialog*( ); *XmTrackingLocate*( ) in the X/Open CAE Specification, **Motif Toolkit API**.

CHANGE HISTORY

First released in Issue 1.

NAME

DtHelpSetCatalogName — assign the name of the message catalogue to use for help services

SYNOPSIS

```
#include <Dt/Help.h>

void DtHelpSetCatalogName(char *catFile);
```

DESCRIPTION

The *DtHelpSetCatalogName*( ) function provides an interface for applications to set the name of the message catalogue file that the help services library uses at runtime. This message catalogue contains all strings, messages and button labels used in the help widgets that can be localised.

The *catFile* argument specifies the name of the message catalogue file that the help services library accesses at runtime. See *catopen*( ) for more information on the message catalogue naming and location semantics for various environments.

RETURN VALUE

The *DtHelpSetCatalogName*( ) function returns no value.

APPLICATION USAGE

The *DtHelpSetCatalogName*( ) function is only required if applications deliver localised online help information into a non-localised XCDE desktop environment. In this case, applications must use this function and give the help message catalogue file a unique name in order to avoid collision with other clients using and localising their online-help user interface. In order for this call to properly affect the help services library, this call must be made prior to creation of any help widgets.

SEE ALSO

<Dt/Help.h>, *DtCreateHelpQuickDialog*( ), *DtCreateHelpDialog*( ); *catopen*( ) in the X/Open CAE Specification, **System Interfaces and Headers**, **Issue 4, Version 2**.

CHANGE HISTORY

First released in Issue 1.

4.4 Headers

This section describes the contents of headers used by the XCDE help service functions, macros and external variables.

Headers contain the definition of symbolic constants, common structures, preprocessor macros and defined types. Each function in Section 4.3 specifies the headers that an application must include in order to use that function. In most cases only one header is required. These headers are present on an application development system; they do not have to be present on the target execution system.

NAME

Dt/Help.h — help services definitions

SYNOPSIS

```
#include <Dt/Help.h>
```

DESCRIPTION

The **<Dt/Help.h>** header defines the variables and function prototypes for help services.

The header defines the following **DtHelpDialogCallbackStruct** *windowHint* constants:

```
DtHELP_POPUP_WINDOW
DtHELP_CURRENT_WINDOW
DtHELP_NEW_WINDOW
```

The header defines the following **DtHelpDialogCallbackStruct** *hyperType* constants:

```
DtHELP_LINK_JUMP_NEW
DtHELP_LINK_TOPIC
DtHELP_LINK_MAN_PAGE
DtHELP_LINK_APP_DEFINE
DtHELP_LINK_TEXT_FILE
```

The header defines the following **DtHelpDialogCallbackStruct** *reason* constants:

```
DtCR_HELP_LINK_ACTIVATE
DtCR_HELP_CLOSE
DtCR_HELP_HELP
```

The header defines the following **DtNScrollBarPolicy** constants:

```
DtHELP_NO_SCROLLBARS
DtHELP_STATIC_SCROLLBARS
DtHELP_AS_NEEDED_SCROLLBARS
```

The header defines the following **DtNhelpType** constants:

```
DtHELP_TYPE_TOPIC
DtHELP_TYPE_STRING
DtHELP_TYPE_MAN_PAGE
DtHELP_TYPE_FILE
DtHELP_TYPE_DYNAMIC_STRING
```

The header defines the following **DtNpaperSize** constants:

```
DtHELP_PAPERSIZE_LETTER
DtHELP_PAPERSIZE_LEGAL
DtHELP_PAPERSIZE_EXECUTIVE
DtHELP_PAPERSIZE_A4
DtHELP_PAPERSIZE_B5
```

The header defines the following *DtHelpQuickDialogGetChild*( ) constants:

```
DtHELP_QUICK_CLOSE_BUTTON
DtHELP_QUICK_PRINT_BUTTON
DtHELP_QUICK_HELP_BUTTON
DtHELP_QUICK_SEPARATOR
DtHELP_QUICK_MORE_BUTTON
DtHELP_QUICK_BACK_BUTTON
```

The header defines the following *DtHelpReturnSelectedWidgetId*( ) constants:

```
DtHELP_SELECT_ERROR
DtHELP_SELECT_VALID
DtHELP_SELECT_ABORT
DtHELP_SELECT_INVALID
```

The header declares the following as functions:

```
void DtHelpSetCatalogName(char *catFile);

int DtHelpReturnSelectedWidgetId(Widget parent,
                                 Cursor cursor,
                                 Widget *widget);
```

CHANGE HISTORY

First released in Issue 1.

NAME

Dt/HelpDialog.h — DtHelpDialog definitions

SYNOPSIS

```
#include <Dt/HelpDialog.h>
```

DESCRIPTION

The **<Dt/HelpDialog.h>** header defines the variables and function prototypes for help dialog services.

The **<Dt/HelpDialog.h>** header declares the following variable:

```
WidgetClass     dtHelpDialogWidgetClass;
```

The header declares the following as a function:

```
Widget DtCreateHelpDialog(Widget parent,
                          char *name,
                          ArgList *arglist,
                          Cardinal argcount);
```

CHANGE HISTORY

First released in Issue 1.

NAME

Dt/HelpQuickD.h — DtHelpQuickDialog definitions

SYNOPSIS

```
#include <Dt/HelpQuickD.h>
```

DESCRIPTION

The **<Dt/HelpQuickD.h>** header defines the variables and function prototypes for help quick dialog services.

The **<Dt/HelpQuickD.h>** header declares the following variable:

```
WidgetClass     dtHelpQuickDialogWidgetClass;
```

The header declares the following as a function:

```
Widget DtCreateHelpQuickDialog(Widget parent,
                               char *name,
                               Arg *arglist,
                               Cardinal argcount);
```

CHANGE HISTORY

First released in Issue 1.

4.5 Actions

This section defines the actions that provide XCDE help services to support application portability at the C-language source or shell script levels.

NAME

dtmanaction — XCDE manual page actions

SYNOPSIS

```
Dtmanpageview [page]
Open page
Print page
```

DESCRIPTION

The XCDE Help Services support the following manual page actions:

Dtmanpageview
Prompt the user for a manual page and open a view of the manual page specified by the user.

Dtmanpageview *page*
Open a view of the manual page named by the pathname in the *page* argument.

Open *page*
Open a view of the manual page named by the pathname in the *page* argument.

Print *page*
Print the manual page named by the pathname in the *page* argument.

These actions can be invoked from an application using the *DtActionInvoke*( ) function or invoked from a command line using the *dtaction* utility.

SEE ALSO

dtaction, *DtActionInvoke*( ) in the **XCDI** specification.

CHANGE HISTORY

First released in Issue 1.

NAME

dthelpaction — XCDE help actions

SYNOPSIS

```
Dthelpview [volume]
Open volume
```

DESCRIPTION

The XCDE Help Services support the following help actions:

Dthelpview
Open a view of the top level help index.

Dthelpview *volume*
Open a view of the help volume named by the pathname in the *volume* argument.

Open *volume*
Open a view of the help volume named by the pathname in the *volume* argument.

These actions can be invoked from an application using the *DtActionInvoke*( ) function or invoked from a command line using the *dtaction* utility.

SEE ALSO

dtaction, *DtActionInvoke*( ) in the **XCDI** specification.

CHANGE HISTORY

First released in Issue 1.

4.6 Formats

HelpTag is a markup language used for authoring XCDE help volumes. It is based on the ISO 8879: 1986 Standard Generalised Markup Language (SGML) standard and is defined with the following Document Type Description (DTD).

The HelpTag marked-up help information is part of the source code for a conforming XCDE application. The compilation process that is required to install this information into help volumes for the XCDE help services is implementation dependent.

```
<!SGML "ISO 8879:1986"

CHARSET
BASESET "ISO 646-1983//CHARSET International Reference Version
          (IRV)//ESC 2/5 4/0"
DESCSET      0        9      UNUSED
             9        2      9
            11        2      UNUSED
            13        1      13
            14       18      UNUSED
            32       95      32
           127        1      UNUSED

BASESET "ISO Registration Number 100//CHARSET ECMA-94
          Right Part of Latin Alphabet  Nr. 1//ESC  2/13  4/1"
DESCSET    128       32      UNUSED
           160        5      32
           165        1      UNUSED
           166       88      38
           254        1      127
           255        1      UNUSED

CAPACITY SGMLREF
    TOTALCAP 350000
    ENTCAP   100000
    ENTCHCAP  50000
    ELEMCAP   50000
    GRPCAP   210000
    EXGRPCAP  50000
    EXNMCAP   50000
    ATTCAP    50000
    ATTCHCAP  50000
    AVGRPCAP  50000
    NOTCAP    50000
    NOTCHCAP  50000
    IDCAP     50000
    IDREFCAP  50000
    MAPCAP   210000
    LKSETCAP  50000
    LKNMCAP   50000

SCOPE DOCUMENT
SYNTAX -- The Core Reference Syntax except with ATTCNT, LITLEN, NAMELEN,
          GRPCNT, and GRPGTCNT changed --

SHUNCHAR CONTROLS   0   1   2   3   4   5   6   7   8   9
                   10  11  12  13  14  15  16  17  18  19
                   20  21  22  23  24  25  26  27  28  29
                   30  31 127 255

BASESET "ISO 646-1983//CHARSET International Reference Version
          (IRV)//ESC 2/5 4/0"
```

```
DESCSET      0      128      0

FUNCTION     RE       13
             RS       10
             SPACE    32
             TAB      SEPCHAR    9

NAMING
   LCNMSTRT ""
   UCNMSTRT ""
   LCNMCHAR "-."
   UCNMCHAR "-."
   NAMECASE
      GENERAL YES
      ENTITY  YES

DELIM
   GENERAL  SGMLREF
   SHORTREF SGMLREF -- Removed short references --
   NAMES    SGMLREF
   QUANTITY SGMLREF
       ATTCNT    140
       LITLEN   4096
       NAMELEN    64
       GRPCNT    100
       GRPGTCNT  253
       TAGLVL     48

FEATURES
    MINIMIZE
        DATATAG    NO
        OMITTAG    NO
        RANK       NO
        SHORTTAG   YES
    LINK
        SIMPLE     NO
        IMPLICIT   NO
        EXPLICIT   NO
    OTHER
        CONCUR     NO
        SUBDOC     NO
        FORMAL     NO
        APPINFO  NONE
>

<!DOCTYPE helpvolume [
<!ELEMENT helpvolume   - - (metainfo?,
                            hometopic?,
                            (chapter* | (s1*, rsect*)),
                            message?,
                            glossary?)
                           +(memo | idx) >

<!ELEMENT metainfo     - - (idsection, abstract?, otherfront*)
                           -(footnote) >

<!ELEMENT idsection    - - (title, copyright?) >

<!ELEMENT title        - - (partext)
                           -(memo | location | idx) >

<!ELEMENT partext      - - ((#PCDATA | acro   | emph     | computer |
                             user    | term   | var      | circle   |
```

```
                                quote   | keycap | graphic  | super    |
                                sub     | book   | xref     | footnote |
                                esc     | link   | location | newline  )*) >

<!ELEMENT acro          - - ((#PCDATA | esc | super | sub)*) >

<!ELEMENT emph          - - (partext) -(emph) >

<!ELEMENT computer      - - ((#PCDATA | quote | var | user | esc)*) >

<!ELEMENT user          - - ((#PCDATA | var | esc)*) >

<!ELEMENT term          - - (partext)
                            -(emph  | computer | term | var     |
                              quote | user     | book | footnote) >
<!ATTLIST term              base         CDATA             #IMPLIED
                            gloss        (gloss | nogloss) gloss >

<!ELEMENT var           - - ((#PCDATA | esc)*) >

<!ELEMENT circle        - - CDATA >

<!ELEMENT quote         - - (partext) -(quote) >

<!ELEMENT keycap        - - ((#PCDATA | super | sub | esc)+) >

<!ELEMENT graphic       - O EMPTY >
<!ATTLIST graphic           id           ID                #IMPLIED
                            entity       ENTITY            #REQUIRED >

<!ELEMENT super         - - (#PCDATA) >

<!ELEMENT sub           - - (#PCDATA) >

<!ELEMENT book          - - (partext) -(book) >

<!ELEMENT xref          - O EMPTY >
<!ATTLIST xref              id           IDREF             #REQUIRED >

<!ELEMENT footnote      - - (p+) -(footnote) >

<!ELEMENT esc           - - CDATA >

<!ELEMENT link          - - (partext) -(link | xref) >
<!ATTLIST link              hyperlink    CDATA             #REQUIRED
                            type         (jump        |
                                          jumpnewview |
                                          definition  |
                                          execute     |
                                          appdefined  |
                                          man         )    jump
                             description CDATA             #IMPLIED >

<!ELEMENT location      - - (partext) -(location) >
<!ATTLIST location          id           ID                #REQUIRED >

<!ELEMENT copyright     - - (text)
                             -(memo | location | idx) >

<!ELEMENT text          - - ((p       | note      | caution   | warning |
                              lablist | list      | ex        | vex     |
                              esc     | otherhead | procedure | syntax  |
                              figure  | image     )* ) >

<!ELEMENT p             - - (head?, partext)
                            +(newline) >
<!ATTLIST (p | image)        indent       (indent)          #IMPLIED
                            id           ID                #IMPLIED
                            gentity      ENTITY            #IMPLIED
```

```
                            gposition    (left | right)    left
                            ghyperlink   CDATA             #IMPLIED
                            glinktype    (jump        |
                                          jumpnewview |
                                          definition  |
                                          execute     |
                                          appdefined  |
                                          man         )    jump
                            gdescription CDATA             #IMPLIED >

<!ELEMENT head          - - (partext)
                            -(memo | location | idx) >

<!ELEMENT newline       - O EMPTY >

<!ELEMENT (note    |
           caution |
           warning )    - - (head?, text)
                             -(note | caution | warning | footnote) >

<!ELEMENT lablist       - - (head?, labheads?, lablistitem+) >
<!ATTLIST lablist           spacing      (loose | tight)   loose
                            longlabel    (wrap  | nowrap)  wrap >

<!ELEMENT labheads      - - (labh, labhtext)
                            -(memo | location | idx) >

<!ELEMENT labh          - - (partext) >

<!ELEMENT labhtext      - - (partext) >

<!ELEMENT lablistitem   - - (label, text) >

<!ELEMENT label         - - (partext) >

<!ELEMENT list          - - (head?, item+) >
<!ATTLIST list              type         (order   |
                                          bullet  |
                                          plain   |
                                          check   )        bullet
                            ordertype    (ualpha  |
                                          lalpha  |
                                          arabic  |
                                          uroman  |
                                          lroman  )        arabic
                            spacing      (tight   |
                                          loose   )        tight
                            continue     (continue)       #IMPLIED >

<!ELEMENT item          - - (text) >
<!ATTLIST item              id           ID               #IMPLIED >

<!ELEMENT ex            - - (head?, (exampleseg, annotation?)+)
                            -(ex      |
                              vex     |
                              note    |
                              caution |
                              warning |
                              syntax  |
                              footnote) >
<!ATTLIST ex                notes        (side | stack)    side
                            lines        (number   |
                                          nonumber )       nonumber
                            textsize     (normal   |
                                          smaller  |
```

```
                                        smallest )       normal >

<!ELEMENT exampleseg    - - (partext) +(lineno) >

<!ELEMENT annotation    - - (partext) +(newline) >

<!ELEMENT lineno        - O EMPTY >
<!ATTLIST lineno            id           ID                #IMPLIED >

<!ELEMENT vex           - - CDATA >
<!ATTLIST vex               lines        (number   |
                                          nonumber )       nonumber
                            textsize     (normal   |
                                          smaller  |
                                          smallest )       normal >

<!ELEMENT otherhead     - - (head, text?) >

<!ELEMENT procedure     - - (chaphead, text?)
                            -(procedure) >

<!ELEMENT chaphead      - - (head, abbrev?)
                            -(memo | location | idx | footnote) >

<!ELEMENT abbrev        - - (partext) -(footnote) >

<!ELEMENT syntax        - - (head?, synel) >

<!ELEMENT synel         - - ((#PCDATA | esc | var |
                              optblock | reqblock  )+) >

<!ELEMENT (optblock |
           reqblock )   - - (synel+) >

<!ELEMENT figure        - - (caption?)
                            -(figure | graphic) >
<!ATTLIST figure            number       NUMBER            #IMPLIED
                            tonumber     (number  |
                                          nonumber)        number
                            id           ID                #IMPLIED
                            entity       ENTITY            #REQUIRED
                            figpos       (left   |
                                          center |
                                          right  )         #IMPLIED
                            cappos       (capleft   |
                                          capcenter |
                                          capright  )      #IMPLIED
                            ghyperlink   CDATA             #IMPLIED
                            glinktype    (jump        |
                                          jumpnewview |
                                          definition  |
                                          execute     |
                                          appdefined  |
                                          man         )    jump
                            gdescription CDATA             #IMPLIED >

<!ELEMENT caption       - - (partext, abbrev?)
                            -(memo | location | idx) >

<!ELEMENT image         - - (head?, partext) -(footnote) >

<!ELEMENT abstract      - - (head?, text?, frontsub*) >

<!ELEMENT frontsub      - - (head?, text) >

<!ELEMENT otherfront    - - (head?, text?, frontsub*) >
<!ATTLIST otherfront        id           ID                #IMPLIED >
```

```
<!ELEMENT hometopic    - - (chaphead, text?) >

<!ELEMENT chapter      - - (chaphead, text?, (s1*, rsect*)) >
<!ATTLIST (chapter |
           s1      |
           s2      |
           s3      |
           s4      |
           s5      |
           s6      |
           s7      |
           s8      |
           s9      )       id           ID                #IMPLIED >

<!ELEMENT s1           - - (chaphead, text?, s2*, rsect*) >

<!ELEMENT s2           - - (chaphead, text?, s3*, rsect*) >

<!ELEMENT s3           - - (chaphead, text?, s4*, rsect*) >

<!ELEMENT s4           - - (chaphead, text?, s5*, rsect*) >

<!ELEMENT s5           - - (chaphead, text?, s6*, rsect*) >

<!ELEMENT s6           - - (chaphead, text?, s7*, rsect*) >

<!ELEMENT s7           - - (chaphead, text?, s8*, rsect*) >

<!ELEMENT s8           - - (chaphead, text?, s9*, rsect*) >

<!ELEMENT s9           - - (chaphead, text?) >

<!ELEMENT rsect        - - (chaphead, text?, rsub*) >
<!ATTLIST rsect           id           ID                #IMPLIED >

<!ELEMENT rsub         - - (chaphead, text?) >

<!ELEMENT message      - - (chaphead?, text?, (msg+ | msgsub+)) >

<!ELEMENT msg          - - (msgnum?, msgtext, explain?) +(newline) >

<!ELEMENT msgnum       - - ((#PCDATA | esc)+) >

<!ELEMENT msgtext      - - (partext) >

<!ELEMENT explain      - - (text) >

<!ELEMENT msgsub       - - (chaphead, text?, msg+) >

<!ELEMENT glossary     - - (text?, glossent+) >

<!ELEMENT glossent     - - (dterm, definition) >

<!ELEMENT dterm        - - (partext) -(term) >

<!ELEMENT definition   - - (text) >

<!ELEMENT idx          - - (indexprimary, indexsub?)
                           -(term | footnote | location | idx) >

<!ELEMENT indexprimary - - (partext, sort?) >

<!ELEMENT indexsub     - - (partext, sort?) >

<!ELEMENT sort         - - ((#PCDATA | esc)+) >

<!ELEMENT memo         - - CDATA >

<!ENTITY MINUS     SDATA "-">
<!ENTITY PM        SDATA '[plusmn]'>    <!-- ISOnum  -->
<!ENTITY DIV       SDATA '[divide]'>    <!-- ISOnum  -->
<!ENTITY TIMES     SDATA '[times ]'>    <!-- ISOnum  -->
<!ENTITY LEQ       SDATA '[le    ]'>    <!-- ISOtech -->
```

```
<!ENTITY GEQ       SDATA '[ge    ]'>    <!-- ISOtech -->
<!ENTITY NEQ       SDATA '[ne    ]'>    <!-- ISOtech -->
<!ENTITY COPY      SDATA '[copy  ]'>    <!-- ISOnum  -->
<!ENTITY REG       SDATA '[reg   ]'>    <!-- ISOnum  -->
<!ENTITY TM        SDATA '[trade ]'>    <!-- ISOnum  -->
<!ENTITY ELLIPSIS  SDATA '[hellip]'>    <!-- ISOpub  -->
<!ENTITY VELLIPSIS SDATA '[vellip]'>    <!-- ISOpub  -->
<!ENTITY PELLIPSIS SDATA "...."> <!-- ellipsis followed by a period -->
<!ENTITY A.M.      SDATA "a.m.">
<!ENTITY P.M.      SDATA "p.m.">
<!ENTITY MINUTES   SDATA '[prime ]'>    <!-- ISOtech -->
<!ENTITY SECONDS   SDATA '[Prime ]'>    <!-- ISOtech -->
<!ENTITY DEG       SDATA '[deg   ]'>    <!-- ISOnum  -->
<!ENTITY SQUOTE    SDATA "'">
<!ENTITY DQUOTE    SDATA '"'>
<!ENTITY ENDASH    SDATA "-">
<!ENTITY EMDASH    SDATA '[mdash ]'>    <!-- ISOpub  -->
<!ENTITY VBLANK    SDATA "_">
<!ENTITY CENTS     SDATA '[cent  ]'>    <!-- ISOnum  -->
<!ENTITY STERLING  SDATA '[pound ]'>    <!-- ISOnum  -->
<!ENTITY SPACE     SDATA " ">
<!ENTITY SIGSPACE  SDATA "& ">
<!ENTITY SIGDASH   SDATA "&-">
<!ENTITY MICRO     SDATA '[micro ]'>    <!-- ISOnum  -->
<!ENTITY OHM       SDATA '[ohm   ]'>    <!-- ISOnum  -->
<!ENTITY UP        SDATA '[uarr  ]'>    <!-- ISOnum  -->
<!ENTITY DOWN      SDATA '[darr  ]'>    <!-- ISOnum  -->
<!ENTITY LEFT      SDATA '[larr  ]'>    <!-- ISOnum  -->
<!ENTITY RIGHT     SDATA '[rarr  ]'>    <!-- ISOnum  -->
<!ENTITY HOME      SDATA "home key">
<!ENTITY BACK      SDATA "<--">
<!ENTITY HALFSPACE SDATA " ">

<!ENTITY % user-defined-entities  SYSTEM "helptag.ent">

%user-defined-entities;

] >
```

4.7 Capabilities

A conforming implementation of the XCDE help services supports at least the following capabilities:

1. Provides both General Help and Quick Help windows with the capabilities described in the following subsections. The General Help window provides access to full navigation capabilities. The Quick Help window provides limited navigation and presentation.
2. Conforms to the Required items in the Application Style Checklist in Chapter 20 on page 355.
3. Has been internationalised using the standard interfaces in the X/Open CAE Specification, **System Interfaces and Headers**, **Issue 4, Version 2**, the X/Open CAE Specification, **Window Management: Xlib C Language Binding**, and the X/Open CAE Specification, **Motif Toolkit API**, and supports any locale available in the underlying X Window System Xlib implementation.

4.7.1 Presentation in the Quick Help Window

The Quick Help window has the capability to present a help topic. Help topics can be authored in HelpTag (specified in Section 4.6 on page 65) or any of the other help types specified in *DtHelpQuickDialog* (see Section 4.2 on page 33).

4.7.2 Navigation in the Quick Help Window

The user controls navigation of help information. The Quick Help window allows the user to:

1. Use scroll bars to see portions of the current topic that do not fit into the current window.
2. Traverse hypertext links.
3. Return to previously viewed topics.
4. Obtain help on using the window.
5. Copy text from the current help topic to another window.
6. Print text from the current help topic.

4.7.3 Presentation for the General Help Window

The General Help window has the capability to present the following:

1. A help topic. Help topics can be authored in HelpTag (specified in Section 4.6 on page 65) or any of the other help types specified in *DtHelpQuickDialog* (see Section 4.2 on page 33).
2. A list of the HelpTag topics of the current help volume that can be browsed.
3. A list of the topics previously visited.
4. A list of the index entries for HelpTag topics of the current volume.

4.7.4 Navigation for the General Help Window

The user controls navigation of help information. The General Help window allows users to:

1. Perform the navigation tasks as specified for the Quick Help window.
2. Go to the beginning of the current help volume.
3. Select an index entry and go to the topic to which the index refers.
4. Access the presentation capabilities listed in Section 4.7.3 on page 72.

Chapter 5

Calendar and Appointment Services

5.1 Introduction

The XCDE calendar and appointment services allow users to schedule appointments and browse or update other users' calendars when trying to set up group meetings. The key supported tasks for the calendar and appointment services are:

- Schedule appointments
- Create ToDo lists
- Schedule repetitive appointments
- Display reminders
- Schedule group appointments
- View a calendar that shows Day, Week, Month or Year time periods
- Multi-browse calendars of multiple users
- Send e-mail with a calendar appointment included to a group
- Print various views of the calendar
- Send e-mail reminders
- Archive calendar data and restore that data

5.2 Functions

The majority of the C-language API to calendar and appointment services is provided by the functions defined in the X/Open CAE Specification, **Calendaring and Scheduling API (XCS)**. All mandatory functions in the **XCS** specification are required on XCDE systems. In addition, the following functions, designated as optional in the **XCS** specification, are required on XCDE systems:

csa_add_calendar( )
csa_call_callbacks( )
csa_list_calendar_attributes( )
csa_list_calendars( )
csa_list_entry_sequence( )
csa_read_calendar_attributes( )
csa_read_next_reminder( )
csa_register_callback( )
csa_unregister_callback( )
csa_update_calendar_attributes( )

The remainder of this section defines an additional function that provides XCDE calendar and appointment services to support application portability at the C-language source level.

NAME

csa_x_process_updates — invoke a calendar application's calendar event handler

SYNOPSIS

```
#include <csa/csa.h>

void csa_x_process_updates(CSA_session_handle cal);
```

DESCRIPTION

The *csa_x_process_updates*( ) function checks to see if there have been calendar updates that are of interest to the client. If there have been one or more calendar updates, and the client previously registered a callback handler using *csa_register_callback*( ) for updates of this type, the callback function is called by *csa_x_process_updates*( ).

The *cal* argument specifies a calendar session handle. The callback function will be invoked only if there have been updates to this calendar. If *cal* is set to NULL, the callback function will be invoked if there have been updates to any logged on calendars.

RETURN VALUE

The *csa_x_process_updates*( ) function returns no value.

APPLICATION USAGE

The *csa_x_process_updates*( ) function is useful for applications that maintain a dynamic representation of calendar information, such as a GUI calendar display. Because the calendar server can simultaneously maintain multiple read/write connections for the same calendar, any data retrieved by a client should be considered immediately out of date. To create the appearance of a dynamic display of calendar data, such an application should call *csa_x_process_updates*( ) as frequently as necessary from within its main event loop.

One way to do this is to have a timeout handler call *csa_x_process_updates*( ) at regular intervals. The duration of the timer should be appropriate for the expected user environment. This does not eliminate the risk of the client holding outdated information; it merely gives the application control over how old the information can get.

SEE ALSO

<csa/csa.h>, *csa_register_callback*( ) in the X/Open CAE Specification, **Calendaring and Scheduling API (XCS)**.

CHANGE HISTORY

First released in Issue 1.

5.3 Headers

The C-language API to calendar and appointment services is provided by the header defined in the X/Open CAE Specification, **Calendaring and Scheduling API (XCS)**. The following header is required on XCDE systems:

<csa/csa.h>

5.4 Command-Line Interfaces

This section defines the utilities that provide XCDE calendar and appointment services.

NAME

dtcm_admin — administer the calendar and appointment services database

SYNOPSIS

```
dtcm_admin [–d] [–a action] [–c calendar] [–s start_date] [–e end_date]
[–f filename]

dtcm_admin –h
```

DESCRIPTION

The *dtcm_admin* utility is the administration interface to the XCDE calendar and appointment services, used to archive and restore data from user calendars.

Archiving calendar information dumps data from the calendar's database within a specified date range, and stores it in an archive file for backup or other purposes.

By default, *dtcm_admin* performs a nondestructive operation, which is useful for backup or data transfer between calendars; the **–d** removes the archived data from the calendar.

OPTIONS

The *dtcm_admin* utility supports the X/Open Utility Syntax Guidelines. The following options are available:

–a *action*
: Specify the operation to be performed. The supported argument values are **archive** and **restore**. The default action is to archive data.

–c *calendar*
: Specify the name of the calendar on which *action* will be performed. Permission to perform the operation is required. Calendar names are implementation-dependent, but typically take the form *user@hostname*, where *user* is a user's login name and *hostname* is the host machine name. If no target calendar is specified, the calendar defaults to the current user at the current host machine.

–d
: Delete the data from the calendar after archiving it. By default the data is not deleted. This only has an effect for archive operations.

–h
: Write a usage help message to standard output and terminate.

–s *start_date*
: Specify the beginning of a date range that constrains the effect of the operation. The *start_date* option-argument is in the form *mmddyy*, where *mm*, *dd* and *yy* are the two-digit month, day and year modulo 100, respectively. The granularity of the range is one day, meaning that the operation is performed on every calendar entry whose start time falls from 00:00 on the specified *start_date* date to 23:59, inclusive, on the *end_date* date. If *start_date* is not specified, all the entries in the calendar dated before or on the *end_date* date are processed. If *end_date* is not specified, all the entries in the calendar dated on or after the *start_date* date are processed. If neither **–s** nor **–e** is specified, the operation is performed for the entire input data set.

–e *end_date*
: Specify the end of a date range that constrains the effect of the operation, in the same format as **–s**.

–f *filename*
: Specify the name of a file for the output of an archive operation or input to a restore operation. If this option is omitted, archived data is sent to standard output and data for a restore operation is taken from the standard input.

OPERANDS

None.

STDIN

When –**a restore** is specified, the standard input is archived data in the format described in Section 5.7.2 on page 92. The standard input is used only if no –**f** *filename* option is specified.

INPUT FILES

When –**a restore** and –**f** *filename* are specified, the input file named by *filename* is archived data in the format described in Section 5.7.2 on page 92.

ENVIRONMENT VARIABLES

The following environment variables affect the execution of *dtcm_admin*:

LANG Provide a default value for the internationalisation variables that are unset or null. If *LANG* is unset or null, the corresponding value from the implementation-specific default locale will be used. If any of the internationalisation variables contains an invalid setting, the utility behaves as if none of the variables had been defined.

LC_ALL If set to a non-empty string value, override the values of all the other internationalisation variables.

LC_MESSAGES Determine the locale that is used to affect the format and contents of diagnostic messages written to standard error and informative messages written to standard output.

NLSPATH Determine the location of message catalogues for the processing of *LC_MESSAGES*.

RESOURCES

None.

ASYNCHRONOUS EVENTS

Default.

STDOUT

Unless –**a restore** and –**f** *filename* are specified, the standard output contains the archived data in the format described in Section 5.7.2 on page 92. If –**f** *filename* is specified, the standard output is not used.

STDERR

Used only for diagnostic messages.

OUTPUT FILES

If –**a archive** and –**f** *filename* are specified, the file named by *filename* contains the archived data in the format described in Section 5.7.2 on page 92.

EXTENDED DESCRIPTION

None.

EXIT STATUS

The following exit values are returned:

0 Successful completion.

>0 An error occurred.

CONSEQUENCES OF ERRORS

Default.

APPLICATION USAGE

None.

EXAMPLES

The following archives calendar data up to the end of 1993:

```
dtcm_admin –a archive –e 123193 –f calendar.1993
```

The following merges another user's calendar data into the user's calendar:

```
dtcm_admin –a archive –c theboss@bigcheese | dtcm_admin –a restore
```

SEE ALSO

dtcm_delete, *dtcm_insert*, *dtcm_lookup*.

CHANGE HISTORY

First released in Issue 1.

NAME

dtcm_delete — delete appointments from the calendar database

SYNOPSIS

`dtcm_delete [–c calendar] [–d date] [–v view]`

DESCRIPTION

The *dtcm_delete* utility is non-GUI interface to the XCDE calendar and appointment services, used to delete appointments from the calendar database. Appointments are deleted one at a time. Each of the components of an appointment is specified using one of the command-line options. The current list of appointments for the specified date (see the **–d** and **–v** options) is displayed, numbered sequentially starting with 1. The user is prompted for the number to delete. Once an appointment is deleted, the list of remaining appointments is redisplayed. At this point the user can specify another number, or just <carriage-return> to quit.

OPTIONS

The *dtcm_delete* utility supports the X/Open Utility Syntax Guidelines. The following options are available:

–c *calendar*
: Specify the name of the target calendar. Calendar names are implementation-dependent, but typically take the form *user@hostname*, where *user* is a user's login name and *hostname* is the host machine name. If no target calendar is specified, the calendar defaults to the current user at the current host machine.

–d *date*
: Specify the date for the appointment(s) to be deleted. The *date* is specified using the form *mm*/*dd*/*yy*, where *mm*, *dd* and *yy* are the two-digit month, day and year modulo 100, respectively. If no date is specified, *date* defaults to today's date.

–v *view*
: Specify the view span of appointments to display. The *view* option-argument can be:

day	Display all appointments for the given date (see **–d** option).
week	Display the full week that contains the given date, starting with Sunday.
month	Display the entire month that contains the given date.

OPERANDS

None.

STDIN

The standard input is used to receive user replies to the list of appointments to be deleted.

INPUT FILES

None.

ENVIRONMENT VARIABLES

The following environment variables affect the execution of *dtcm_delete*:

LANG
: Provide a default value for the internationalisation variables that are unset or null. If *LANG* is unset or null, the corresponding value from the implementation-specific default locale will be used. If any of the internationalisation variables contains an invalid setting, the utility behaves as if none of the variables had been defined.

LC_ALL If set to a non-empty string value, override the values of all the other internationalisation variables.

LC_MESSAGES Determine the locale that is used to affect the format and contents of diagnostic messages written to standard error and informative messages written to standard output.

NLSPATH Determine the location of message catalogues for the processing of *LC_MESSAGES*.

RESOURCES

None.

ASYNCHRONOUS EVENTS

Default.

STDOUT

The standard output contains the list of appointments to be deleted, in an unspecified format.

STDERR

Used only for diagnostic messages.

OUTPUT FILES

None.

EXTENDED DESCRIPTION

None.

EXIT STATUS

The following exit values are returned:

0 Successful completion.

>0 An error occurred.

CONSEQUENCES OF ERRORS

Default.

APPLICATION USAGE

None.

EXAMPLES

None.

SEE ALSO

dtcm_admin, *dtcm_insert*, *dtcm_lookup*.

CHANGE HISTORY

First released in Issue 1.

NAME

dtcm_insert — insert appointments into the calendar database

SYNOPSIS

`dtcm_insert [–c` *`calendar`*`] [–d` *`date`*`] [–s` *`start`*`] [–e` *`end`*`] [–v` *`view`*`] [–w` *`what`*`]`

DESCRIPTION

The *dtcm_insert* utility is non-GUI interface to the XCDE calendar and appointment services, used to add new appointments to the calendar database. Appointments are added one at a time. Each of the components of an appointment is specified using one of the command-line options. Once an appointment is added, the list of appointments for the specified date (see the **–d** and **–v** options) is displayed.

OPTIONS

The *dtcm_insert* utility supports the X/Open Utility Syntax Guidelines. The following options are available:

–c *calendar*

Specify the name of the target calendar. Calendar names are implementation-dependent, but typically take the form *user@hostname*, where *user* is a user's login name and *hostname* is the host machine name. If no target calendar is specified, the calendar defaults to the current user at the current host machine.

–d *date*

Specify the date for the appointment(s) to be inserted. The *date* is specified using the form *mm*/*dd*/*yy*, where *mm*, *dd* and *yy* are the two-digit month, day and year modulo 100, respectively. If no date is specified, *date* defaults to today's date.

–s *start*

Specify the starting time for the appointment. The time is specified using the form *hh*:*mm*. If *hh* is greater than 12, 24-hour convention (for example, **15:30** instead of **3:30 pm**) is assumed. If *hh* is 0 to 12, an optional **am** or **pm** suffix can be used, with or without white space separating the suffix from the *mm*. If no suffix is used, **am** is assumed. If no starting time is specified, no time is associated with the inserted appointment.

–e *end*

The ending time for the appointment, in the same format as **–s**. Specifying an ending time without a starting time is an error.

–v *view*

Specify the view span of appointments to display. The *view* option-argument can be:

day	Display all appointments for the given date (see **–d** option).
week	Display the full week that contains the given date, starting with Sunday.
month	Display the entire month that contains the given date.

–w *what*

Specify the appointment description text. Up to 5 lines of text can be specified by placing \n (the literal characters \ and n, not <newline>) between lines. If not specified, *what* defaults to **Appointment**.

OPERANDS

None.

STDIN

Not used.

INPUT FILES

None.

ENVIRONMENT VARIABLES

The following environment variables affect the execution of *dtcm_insert*:

LANG Provide a default value for the internationalisation variables that are unset or null. If *LANG* is unset or null, the corresponding value from the implementation-specific default locale will be used. If any of the internationalisation variables contains an invalid setting, the utility behaves as if none of the variables had been defined.

LC_ALL If set to a non-empty string value, override the values of all the other internationalisation variables.

LC_MESSAGES Determine the locale that is used to affect the format and contents of diagnostic messages written to standard error and informative messages written to standard output.

NLSPATH Determine the location of message catalogues for the processing of *LC_MESSAGES*.

RESOURCES

None.

ASYNCHRONOUS EVENTS

Default.

STDOUT

The standard output contains the list of appointments for the specified view span, including the appointment just inserted, in an unspecified format.

STDERR

Used only for diagnostic messages.

OUTPUT FILES

None.

EXTENDED DESCRIPTION

None.

EXIT STATUS

The following exit values are returned:

0 Successful completion.

>0 An error occurred.

CONSEQUENCES OF ERRORS

Default.

APPLICATION USAGE

None.

EXAMPLES

None.

SEE ALSO

dtcm_admin, *dtcm_delete*, *dtcm_lookup*.

CHANGE HISTORY

First released in Issue 1.

NAME

dtcm_lookup — look up appointments from the calendar database

SYNOPSIS

`dtcm_lookup [–c calendar] [–d date] [–v view]`

DESCRIPTION

The *dtcm_lookup* utility is non-GUI interface to the XCDE calendar and appointment services, used to look up appointments from the calendar database. Each component of the calendar entry is specified using one of the command-line options. The current list of appointments for the specified date (see the **–d** and **–v** options) is displayed.

OPTIONS

The *dtcm_lookup* utility supports the X/Open Utility Syntax Guidelines. The following options are available:

–c *calendar*
: Specify the name of the target calendar. Calendar names are implementation-dependent, but typically take the form *user@hostname*, where *user* is a user's login name and *hostname* is the host machine name. If no target calendar is specified, the calendar defaults to the current user at the current host machine.

–d *date*
: Specify the date for the look up query. The *date* is specified using the form *mm*/*dd*/*yy*, where *mm*, *dd* and *yy* are the two-digit month, day and year modulo 100, respectively. If no date is specified, *date* defaults to today's date.

–v *view*
: Specify the view span of appointments to display. The *view* option-argument can be:

  day Display all appointments for the given date (see **–d** option).

  week Display the full week that contains the given date, starting with Sunday.

  month Display the entire month that contains the given date.

OPERANDS

None.

STDIN

Not used.

INPUT FILES

None.

ENVIRONMENT VARIABLES

The following environment variables affect the execution of *dtcm_lookup*:

LANG Provide a default value for the internationalisation variables that are unset or null. If *LANG* is unset or null, the corresponding value from the implementation-specific default locale will be used. If any of the internationalisation variables contains an invalid setting, the utility behaves as if none of the variables had been defined.

LC_ALL If set to a non-empty string value, override the values of all the other internationalisation variables.

LC_MESSAGES Determine the locale that is used to affect the format and contents of diagnostic messages written to standard error and informative messages written to standard output.

NLSPATH Determine the location of message catalogues for the processing of *LC_MESSAGES*.

RESOURCES

None.

ASYNCHRONOUS EVENTS

Default.

STDOUT

The standard output contains the list of appointments for the specified view span, in an unspecified format.

STDERR

Used only for diagnostic messages.

OUTPUT FILES

None.

EXTENDED DESCRIPTION

None.

EXIT STATUS

The following exit values are returned:

0 Successful completion.

>0 An error occurred.

CONSEQUENCES OF ERRORS

Default.

APPLICATION USAGE

None.

EXAMPLES

None.

SEE ALSO

dtcm_admin, *dtcm_insert*, *dtcm_delete*.

CHANGE HISTORY

First released in Issue 1.

5.5 Actions

This section defines the actions that provide XCDE calendar and appointment services to support application portability at the C-language source or shell script levels.

NAME

dtcalendaraction — XCDE calendar and appointment management actions

SYNOPSIS

```
Dtcm
DtcmEdit appointment
DtcmInsert appointment
Open appointment
Insert appointment
```

DESCRIPTION

The XCDE Calendar and Appointment Services support the following calendar and appointment management actions:

Dtcm
Open a view of the user's default calendar.

DtcmEdit *appointment*
Edit the appointment named by the pathname in the *appointment* argument.

DtcmInsert *appointment*
Insert the appointment named by the pathname in the *appointment* argument into the user's default calendar.

Open *appointment*
Edit the appointment named by the pathname in the *appointment* argument.

Insert *appointment*
Insert the appointment named by the pathname in the *appointment* argument into the user's default calendar.

These actions can be invoked from an application using the *DtActionInvoke*( ) function or invoked from a command line using the *dtaction* utility.

All *appointment*s are text files including calendar entries defined in Section 5.7.2 on page 92.

SEE ALSO

dtaction, *DtActionInvoke*( ) in the **XCDI** specification.

CHANGE HISTORY

First released in Issue 1.

5.6 Messages

The XCDE calendar and appointment services implement the *Display* and *Edit* requests for media type _DT_CM_APPOINTMENT. See the **XCDI** specification, **Section 6.6.2**, **Media Exchange Message Set**.

5.7 Formats

5.7.1 Calendar Archive File Format

The calendar archive file format is a container with a series of packages of attribute value triples embedded within it. Each package of attribute/value triples is the definition of an entry or the definition of calendar attributes. A grammar for the archive file is:

```
archive_file      ::= header_string content_list
header_string     ::= 'DTCM Archive 1.0\n'
content_list      ::= package content_list | package
package           ::= appt_def | calendar_attr_def
appt_def          ::= appt_start_marker attr_list appt_end_marker
calendar_attr_def ::= cal_start_marker attr_list cal_end_marker
attr_list         ::= attr_def att_list | attr_def
appt_start_marker ::= appt_marker ':string:begin\n'
appt_end_marker   ::= appt_marker ':string:end\n'
appt_marker       ::=
  '-//CDE_XAPIA_PRIVATE/CS/API/ENTRYATTR/ /NONSGML ApptDelimiter//EN'
cal_start_marker  ::= cal_marker ':string:begin\n'
cal_end_marker    ::= cal_marker ':string:end\n'
cal_marker        ::=
   '-//CDE_XAPIA_PRIVATE/CS/API/CALATTR/ /NONSGML CalDelimiter//EN'
attr_def          ::= attr_name ':' attr_type ':' attr_value '\n'
attr_type         ::= 'string' | 'integer' | 'reminder' | 'accesslist'
                    | 'repeatdefinition' | 'custom'
attr_value        ::= The data associated with the attribute
attr_name         ::= XAPIA string. The span of characters that can be used in the
                      definition of an attribute name is specified in the X/Open CAE
                      Specification, Calendaring and Scheduling API (XCS).
```

An example calendar file with two entries is as follows.

```
DTCM Archive 1.0
-//CDE_XAPIA_PRIVATE/CS/API/ENTRYATTR/ /NONSGML ApptDelimiter//EN:string:begin
attr_name0:type:value
attr_name1:type:value
       ...
-//CDE_XAPIA_PRIVATE/CS/API/ENTRYATTR/ /NONSGML ApptDelimiter//EN:string:end
       .
       .
       .
-//CDE_XAPIA_PRIVATE/CS/API/ENTRYATTR/ /NONSGML ApptDelimiter//EN:string:begin
attr_name0:type:value
attr_name1:type:value
       ...
-//CDE_XAPIA_PRIVATE/CS/API/ENTRYATTR/ /NONSGML ApptDelimiter//EN:string:end
```

5.7.1.1 Attribute Definition

Each entry in a calendar is represented by a series of triples. These triples are constructed of three objects: the attribute name, the type of the attribute and the actual value of the attribute. The attribute name is the same name used to store the attribute in the database and is constructed out of printable characters within the referenced ISO/IEC 8859-1:1987 standard character set. The set of valid characters does not include NUL, <tab>, <newline>, <carriage return>, <space> or colon. The attribute type describes the type of the data associated with the attribute name. This is a limited set of types that includes integer, string, reminder and access list, and buffer. The value is the actual value associated with the attribute name, and is interpreted according to the type value associated with it. Within a calendar archive file, an individual attribute is written with the members of the triple separated by colons:

attrname:*type*:*value*

An example attribute is:

```
-//XAPIA/CS/API/CALATTR/ /NONSGML Calendar Name//EN:
        string:fred@host.Company.COM
```

Attribute types can have the following values. The ''integer/string pairs'' used frequently in this list refer to an **integer** value followed by a colon and a **string** value.

string A null-terminated sequence of characters. The bytes in an attribute of this type are interpreted relative to the character set attribute for the same entry.

integer A decimal integer, expressed as digits from the referenced ISO/IEC 8859-1:1987 standard.

reminder An integer/string pair representation of the reminder structure defined in the X/Open CAE Specification, **Calendaring and Scheduling API (XCS)**. The integer is the advance on the reminder, expressed as a number of seconds, with negative numbers indicating time prior to the event. The string describes additional data that may relate to the reminder. The XCDE calendar and appointment services use this string as a list of e-mail addresses for e-mail reminders, but ignore it for other reminder types. An example entry is:

```
-//XAPIA/CS/API/ENTRYATTR/ /NONSGML Mail Reminder//EN:
        reminder:-1800:hseldon@trantor
```

accesslist A string that describes the individuals who have specific access permissions set on a calendar. This attribute type applies only to a calendar, not an entry, where it is ignored. An access list value is formatted across a series of lines, each line containing one logical access list entry. The first entry starts a line and is indented with a <tab> character. Each line is an integer/string pair. The integer represents the access granted to the user, as defined by the X/Open CAE Specification, **Calendaring and Scheduling API (XCS)**. The string describes the user who gains that access. An example entry is:

```
-//XAPIA/CS/API/CALATTR/ /NONSGML Access List//EN:accesslist:
        11:fred
        7:joanne
```

repeatdefinition
A string representation of a data structure that describes how an event repeats indefinitely. This structure is defined by the X/Open CAE Specification, **Calendaring and Scheduling API (XCS)**.

custom Custom entries are for attributes values that do not conform to the constraints of a string data type because they can have embedded NUL characters within them. It is the responsibility of the application to ensure that these values are portable between systems because the XCDE calendar and appointment services do not perform any transformation on the characters.

5.7.1.2 Long Values

When an attribute value is long, or contains embedded <newline>s or <carriage return>s, the values for attributes are broken out across a number of lines, using a subset of the MIME RFC rules for long/binary headers. The XCDE calendar and appointment services support unbroken lines of at least 256 bytes. All continuation lines begin with a <tab>. Lines that are too long are broken with a <newline>/<tab> pair. Thus, abcd becomes ab\n\tcd. Embedded <newline> are suffixed with a <tab>. Thus, ab\ncd becomes ab\n\tcd.

5.7.2 Calendar Entry Format

The calendar entry file format defines how one entry is saved to a file or used in a drag-and-drop transaction with another client. A grammar for the entry format is:

```
appt_file          ::= header_string entry_definition
                   ::= '\n\n\t**Calendar Appointment **\n'
entry_definition   ::= appt_def '\n' old_appt
                    |  old_appt
old_appt           ::= date_mark start_mark end_mark repeat_mark
                    |  duration_mark text_mark
date_mark          ::= '\tDate: ' date_value '\n'
date_value         ::= Date as mm/dd/yyyy
start_mark         ::= '\tStart: ' start_value '\n'
start_value        ::= Entry begin time as hh:mm[ampm], where hh and mm represent two-digit
                       hours and minutes and the optional ampm suffix is the string am or pm
end_mark           ::= '\tEnd: ' end_value '\n'
end_value          ::= Entry begin time as hh:mm[ampm]
repeat_mark        ::= '\tRepeat: ' repeat_value '\n'
repeat_value       ::= 'One Time' | 'Daily' | 'Weekly' | 'Every Two Weeks'
                    |  'Monthly By Date' | 'Yearly' | 'Monthly By Weekday'
                    |  'Monday Thru Friday' | 'Mon, Wed, Fri'
                    |  'Tuesday, Thursday'
duration_mark      ::= '\tFor: ' duration_value '\n'
duration_value     ::= Integer that describes the number of repetitions.
text_mark          ::= '\tWhat: ' text_value
text_value         ::= Up to 5 lines of text.
                       Each line after the first must have a leading
                       <tab> character.
appt_def           ::= appt_start_marker attr_list appt_end_marker
attr_list          ::= attr_def att_list
                    |  attr_def
appt_start_marker  ::= appt_marker ':string:begin\n'
appt_end_marker    ::= appt_marker ':string:end\n'
appt_marker        ::=
    '-//CDE_XAPIA_PRIVATE/CS/API/ENTRYATTR/ /NONSGML ApptDelimiter//EN'
```

```
attr_def          ::= attr_name ':' attr_type ':' attr_value '\n'
attr_type         ::= 'string' | 'integer' | 'reminder'
                    | 'accesslist' | 'repeatdefinition' | 'custom'
attr_name         ::= XAPIA string. The span of characters that can be used in the definition
                      of an attribute name is specified in the X/Open CAE Specification,
                      Calendaring and Scheduling API (XCS).
```

An example entry:

```
        ** Calendar Appointment **
-//CDE_XAPIA_PRIVATE/CS/API/ENTRYATTR/ /NONSGML ApptDelimiter//EN:string:begin
-//XAPIA/CS/API/ENTRYATTR/ /NONSGML Start Date//EN:integer:775148400
-//XAPIA/CS/API/ENTRYATTR/ /NONSGML End Date//EN:integer:775148900
-//XAPIA/CS/API/ENTRYATTR/ /NONSGML Type//EN:string:CSA_TYPE_EVENT
-//CDE_XAPIA_PRIVATE/CS/API/ENTRYATTR/ /NONSGML Showtime//EN:integer:1
-//XAPIA/CS/API/ENTRYATTR/ /NONSGML Summary//EN:string:Foundation planning
                                          meeting in Hari's office
-//XAPIA/CS/API/ENTRYATTR/ /NONSGML Status//EN:integer:0
-//XAPIA/CS/API/ENTRYATTR/ /NONSGML Recurrence Rule//EN:string:M60 #12
-//XAPIA/CS/API/ENTRYATTR/ /NONSGML Audio Reminder//EN:reminder:-60:
-//XAPIA/CS/API/ENTRYATTR/ /NONSGML Flashing Reminder//EN:reminder:-60:
-//XAPIA/CS/API/ENTRYATTR/ /NONSGML Mail Reminder//EN:reminder:-1800:
                                                 hseldon@trantor
-//XAPIA/CS/API/ENTRYATTR/ /NONSGML Popup Reminder//EN:reminder:300:
-//CDE_XAPIA_PRIVATE/CS/API/ENTRYATTR/ /NONSGML ApptDelimiter//EN:string:end

        Date: 7/25/1994
        Start:  8:00am
        End:  9:00am
        Repeat: Every Two Weeks, last
        For:26
        What: Foundation planning meeting
               in Hari's office
```

5.8 Capabilities

A conforming implementation of the XCDE calendar and appointment services supports at least the following capabilities:

1. Provides calendar and appointment services as described in the following subsections.
2. Conforms to the Required items in the Application Style Checklist in Chapter 20 on page 355.
3. Has been internationalised using the standard interfaces in the X/Open CAE Specification, **System Interfaces and Headers**, **Issue 4, Version 2**, the X/Open CAE Specification, **Window Management: Xlib C Language Binding**, and the X/Open CAE Specification, **Motif Toolkit API**, and supports any locale available in the underlying X Window System Xlib implementation.

5.8.1 Calendar Main Window

The main window of the XCDE calendar and appointment services provides access to all calendar and appointment services functionality and selects the desired calendar view. At least the following views are supported:

Day The appointments for a single work day are displayed within subdivisions. (The times of a work day can be specified as a user option; see Section 5.8.2 on page 95.) Also within the view are three monthly calendars: the preceding, current and following months relative to the current Day view. Choosing a day in this section changes the current work day of appointments being viewed.

Week The appointments for a full week are displayed within subdivisions, each representing one day. Also within the view is a display that represents graphically which times during the week are free, are filled by appointments and are filled by overlapping appointments. Invoking the default action on the graphical display section opens the appointment editor, set to the appropriate day and time.

Month The appointments for a full month are displayed within subdivisions, each representing one day. The display resembles a wall calendar, with multiple rows of seven boxes, arranged into weeks.

Year All of the days of the current year are displayed, by month. Each month display resembles a wall calendar, with multiple rows of seven boxes, arranged into weeks. Appointments need not be displayed. Invoking the default action on a month display may open the appointment editor, set to a day and time within that month.

For the Day, Week and Month views, the following rules apply to appointment displays:

- The appointment text is associated with the start and stop times.
- When viewed by another user, the appointment text may be suppressed, depending on privacy options, described later.
- The number of appointments viewable may be limited by the size of the subdivision.
- Invoking the default action on an appointment subdivision opens the appointment editor, set to the appropriate day and time.

Each view has navigation buttons (in addition to any control available in the window menu) to change the date range in the view: arrow buttons and a Today button. The action of the navigation buttons depends on the current view (Day, Week, Month or Year). For example, in the Week view, the user can click on the arrows on either side of the current day to go backward

or forward one week, and can return to the view of the current week by clicking on the button that displays the current day.

From any view, the user can enter a specific date and go directly to the period of time that includes that date. (For example, in Week view, entering a date will move to the week that contains that date.)

5.8.2 Options/Properties

The following options or properties can be set and they persist through each session:

- The default calendar displayed at startup. This string may be of the form *user@hostname*, or another implementation-dependent format.
- The default calendar view displayed at startup.
- The time the work day starts and ends.
- The access list for the user's calendar. The possible permissions include browse, insert and delete. Access can be granted on a individual user basis or for all other users. The default is that all other users can browse the calendar, but they do not have insert or delete access to it.
- Defaults for appointments: reminder methods (including beep, flash, popup and e-mail) and preceding time; privacy.

5.8.3 Appointment Editing

The appointment editor may be a separate dialog box. The system need not support more than one appointment editor open at any one time.

The user can use the appointment editor to accomplish the following:

1. Add new appointments. Appointments have at least the following attributes:
    a. Date.
    b. Start and stop times, both within that date.
    c. Descriptive text, of at least four lines. This text may be truncated in some calendar views.
    d. Reminders. The user can select one or more of at least the following reminder methods, each with an amount of time specified in advance of the appointment: sounding an audible alarm, if supported by the hardware; flashing the calendar icon and window; displaying a message dialog with the information concerning the appointment; sending an e-mail.
    e. Frequency. The user can schedule repetitive appointments by selecting the number of repetitions and the frequency:
        - once (default)
        - daily
        - weekly on the same day
        - every two weeks on the same day
        - monthly on the same date
        - monthly by weekday

- annually on the same date
- daily on Monday to Friday only
- every Monday, Wednesday and Friday
- every Tuesday and Thursday
- every *n* days, weeks or months, where *n* is an integer specified by the user

f. Privacy. The user can choose to suppress the descriptive text, or all knowledge of this appointment, from other users who have browse access to the calendar.

2. Display a list of all appointments for the day.
3. Delete existing appointments.
4. Edit attributes of existing appointments.

5.8.4 Appointment Listing

The appointment lister may be a separate dialog box. The system need not support more than one appointment lister open at any one time.

The appointment lister lists of appointments for the calendar view at the time the list is opened; for example, in Week view, the list displays all appointments for the current week. Changing the calendar view may change an open appointment list.

The list includes at least the following for each appointment: date; start time; the first line of the appointment description. Appointments are sorted in ascending order by start date and time.

Invoking the default action on an appointment in the list opens the appointment editor, with that appointment selected for editing.

5.8.5 Appointment Finding

The XCDE calendar and appointment services can search for appointments meeting user-specified criteria:

1. The user can specify a string on which to search. The search is independent of case.
2. The user can limit the search to dates within a specified range.

All matching appointments are presented to the user. Selecting one of the matches changes the main window view to include the date of that appointment.

5.8.6 To-Do Editing

The to-do editor may be a separate dialog box. The system need not support more than one to-do editor open at any one time.

The user can use the to-do editor to accomplish the following:

1. Add new to-do tasks. To-do tasks have at least the following attributes:

   a. Optional due date.

   b. Descriptive text, of at least four lines. This text may be truncated in some calendar views.

   c. Status: completed, pending or overdue. Pending or overdue are in relation to the due date. The user can change a pending or overdue task to completed, or remove the completed status.

d. Reminders. This attribute is the same as for an appointment, except that only e-mail reminders need to be supported.

e. Frequency. This attribute is the same as for an appointment.

2. Delete existing to-do tasks. (The task is selected from the to-do list described in the next section.)

3. Edit attributes of existing to-do tasks. (The task is selected from the to-do list described in the next section.)

5.8.7 To-Do Listing

The to-do lister may be a separate dialog box. The system need not support more than one to-do lister open at any one time.

The user can limit the displayed list of to-do tasks to only completed tasks or to pending and overdue tasks.

The user can limit the displayed list of to-do tasks to those tasks due in the current day, week, month or year. To-do tasks that do not have a due date appear in all views.

The displayed list includes at least the following for each to-do task: status; due date; the first line of the task description.

Invoking the default action on a task in the list opens the to-do editor, with that task selected for editing.

5.8.8 Multi-User Calendar Accessing

The user can identify calendars (such as with the *user@hostname* notation) to be in a list of frequently accessed calendars. The user can add and delete calendars from the list. The list contents persist through each session.

The user can access some or all of the calendars from those in the frequently accessed calendar list, to identify common free times. A graphical display depicts time periods in which conflicts occur and the user can determine which calendars are busy for those periods of time. The user can access a XCDE mail services Compose window (see Chapter 6 on page 99) that is pre-addressed to all users whose calendars are selected.

The user can schedule an appointment on all accessible calendars from the preceding step and optionally e-mail the appointment to those not accessible. The user can access a XCDE mail services Compose window (see Chapter 6 on page 99) that is pre-addressed to all users whose calendars are selected and the text of the mail message contains the new or edited appointment.

5.8.9 Drag and Drop Capabilities

The XCDE calendar and appointment services provide drag and drop capabilities as follows:

1. The user can drag a calendar appointment from an icon in the appointment editor or from any list of appointments to any drop site registered to accept buffer drops.

2. The user can drop a buffer containing an appointment onto the main calendar window, causing the appointment to be scheduled.

The format of the appointment to be dragged and dropped is described in Section 5.7.2 on page 92.

5.8.10 Printing

The following printing capabilities are supported:

1. From any view in the main window, the user can print a version of that view.
2. From any view in the main window, the user can print a version of any other view, without changing the view being displayed.
3. The user can print any main view, the appointment list or to-do list and specify a range of times, to be printed as if individual view-printing requests had been made. For example, in Week view, two or more weeks can be printed in a single request.

5.8.11 Other Capabilities

The following additional capabilities are supported:

1. The user can change the calendar display to that of another user, allowing direct browsing or updating (given appropriate access permissions). Each calendar identified in the list of frequently accessed calendars, described in Section 5.8.8 on page 97, is available directly as a menu item. The user can also change to an individual calendar without adding it to the list.
2. The user can change the view of appointments and to-do tasks so that they appear to be in a specified time zone other than the local one.

Chapter 6

Mail Services

6.1 Introduction

The XCDE mail services provide a GUI for manipulating electronic mail messages that may have attachments. Users can use the interface to perform activities such as: compose a message, view the contents of a message, view the list of messages in a mailbox, copy or move messages from one mailbox to another, delete messages, reply to messages, add/delete attachments to a message, and view contents of attachments in a message. The key supported tasks for the mail services are:

- Receive and view mail messages and their attachments
- Compose, reply, forward, reply-include and send messages
- Include files and add attachments to outgoing messages
- File incoming messages in different mailboxes
- Create, open and close mailboxes
- Move or copy messages from one mailbox to another
- Delete and undelete messages
- Print messages
- Sort or find messages based on prescribed criteria
- Provide a public ToolTalk API to support a mail-pervasive desktop environment

6.2 Actions

This section defines the actions that provide XCDE mail services to support application portability at the C-language source or shell script levels.

NAME

dtmailaction — XCDE mail actions

SYNOPSIS

```
Compose [file ...]
Dtmail [file]
Open file
Print file
```

DESCRIPTION

The XCDE Mail Services support the following mail actions:

Compose
Open an empty mail composition view for message construction.

Compose *file* ...
Open a mail composition view with attachments named by the pathnames in the *file* arguments.

Dtmail
Open a view of the user's inbox for electronic mail.

Dtmail *file*
Open a view of the mail file named by the pathname in the *file* argument.

Open *file*
Open a view of the mail file named by the pathname in the *file* argument.

Print *file*
Print the mail file named by the pathname in the *file* argument.

These actions can be invoked from an application using the *DtActionInvoke*( ) function or invoked from a command line using the *dtaction* utility.

SEE ALSO

dtaction, *DtActionInvoke*( ) in the **XCDI** specification.

CHANGE HISTORY

First released in Issue 1.

6.3 Messages

The mail services implement the *Display* and *Mail* media messages for media type RFC_822_MESSAGE. See the **XCDI** specification, **Section 6.6.2**, **Media Exchange Message Set**.

When a *Display* message is received, the mail services display the mailbox specified by the *Display* message's *file* attribute.

When a *Mail*(TT_IN) message is received, the mail services attempt to send the given data.

When a *Mail*(TT_OUT) message is received, the mail services open an empty Compose window.

When a *Mail*(TT_INOUT) message is received, the mail services parse the given data and open a Compose window with the data filled into the appropriate areas (header fields, text and attachment areas).

For *Mail*(TT_IN) and *Mail*(TT_INOUT) messages, the data must be a fully constructed mail message in the format described in Section 6.4. If the data has attachments, it must be in the format described in the referenced MIME RFCs.

6.4 Formats

The XCDE mail services transmit and receive mail messages formatted in accordance with the referenced RFC-822; some or all of the following header fields, as defined in the referenced MIME RFCs, are also included in each message:

```
Content-Description
Content-ID
Content-Transfer-Encoding
Content-Type
Mime-Version
```

The XCDE mail services read and write mailboxes in the format described in the *mailx* command in the X/Open CAE Specification, **Commands and Utilities**, **Issue 4, Version 2**, with each mail message formatted as described previously, except that an additional header line is included in each message:

```
Content-Length: n
```

where *n* is an integer representing the number of octets in the body of the message (in other words, those octets that occur after the empty line separating the message header from the message body). When a mailbox is being read, the Content-Length fields of the messages are verified. If the content length points exactly to the beginning of a **From** line that denotes the start of a new mail message, or if it points to the end of the mailbox file, the Content-Length value is honoured; otherwise, the mail services scan the mailbox for the next message and correct the value of the Content-Length header when the mailbox is later written.

When messages are saved in a mailbox file, the XCDE mail services may write additional header lines into each mail message stored in the file.

6.5 Capabilities

A conforming implementation of the XCDE mail services supports at least the following capabilities:

1. Provides mail services as described in the following subsections.
2. Formats mail messages and mailbox files as described in Section 6.4 on page 101.
3. Conforms to the Required items in the Application Style Checklist in Chapter 20 on page 355.
4. Has been internationalised using the standard interfaces in the X/Open CAE Specification, **System Interfaces and Headers**, **Issue 4, Version 2**, the X/Open CAE Specification, **Window Management: Xlib C Language Binding**, and the X/Open CAE Specification, **Motif Toolkit API**, and supports any locale available in the underlying X Window System Xlib implementation.

6.5.1 Managing Mailboxes

Mailboxes are text files formatted as described in Section 6.4 on page 101. The user has a mailbox referred to as the Inbox where incoming mail is received. The user can also have additional mailboxes. (Unless otherwise noted, the term *mailbox* applies to both the Inbox and any additional mailbox files.) The following capabilities are supported for managing mailboxes:

1. The user can open the Inbox and any additional mailboxes for which he or she has read permission. Multiple mailboxes can be open simultaneously. Mailboxes for which the user has read permission, but does not have write permission, are opened as read-only mailboxes, which prevents deletion of messages.
2. The user can create new, empty mailboxes.
3. The user can close mailboxes.
4. The user can set the method by which the mail services update a mailbox's list of mail messages to account for newly received, moved or copied messages. This can be a recurring time period as well as an immediate one-time update.
5. The user can select audible alarms or flashing windows as a means of announcing that the mailbox has been updated.
6. The user can specify a default directory for mailboxes (other than the Inbox).

6.5.2 Managing Message Lists

The following capabilities are supported for viewing and manipulating the message list associated with a mailbox:

1. When a mailbox is open, it presents a list of its constituent mail messages. The list contains at least the following for each message:
    a. Name or login name of sender
    b. Subject (which may be empty)
    c. Date and time the message was sent, if that information is available in the message, expressed in the time zone of the local system
    d. Size of the message in octets; messages larger than 1 024 octets may be expressed in terms of ''K'' (1 024) octets

e. Status of the message (new or already read)

f. Whether there are attachments to the message

2. The user can delete messages from the list. The deleted message is placed on a list of deleted message and is not physically removed from the mailbox at that time.

3. The user can restore deleted messages by selecting them from the deleted message list.

4. The user can physically remove all deleted messages from the mailbox by destroying the deleted message list. This destruction can occur when the user chooses a control to do such and it can be set up to occur automatically when the mailbox is closed. The user may be offered the choice of removing individual mail messages from the deletion list based on the age of the message or other user-selected criteria.

5. The user can sort the list of messages in any of the following sequences:

   a. Date and time of sending — sorted in date/time sequence

   b. Name or login name of sender — sorted in case-sensitive character sequence

   c. Subject — sorted in case-sensitive character sequence, with replies to messages sorted as if the leading characters ''Re: '' were not present

   d. Size — sorted in size sequence

   e. Status — sorted in an unspecified sequence

6. The user can search for messages in the list, selecting either the next message meeting the criteria or all of the messages doing so, at the user's option. The search matches substrings of any or all of the following header values, in a case-insensitive manner: To, From, Subject, Cc.

7. The user can select one or more messages and copy them to another mailbox. All of a message, including attachments, is copied.

8. The user can select one or more messages and move them to another mailbox. The movement is equivalent to a copy operation followed by a delete operation.

9. The user can select one or more messages and print them. The data transmitted for printing of each message includes the text and mail header of the message. The user may be able to choose to print attachments along with the text and header. The method used to print the text is based on the **Print** action for the DTMAIL_FILE file type; see the **XCDI** specification, **Section 8.4**, **Data Formats**.

6.5.3 Viewing and Manipulating Existing Messages

The following capabilities are supported for viewing and manipulating existing messages in a mailbox:

1. The user can display a message. The user can choose to display additional messages by either replacing the current display with the new message or opening additional display windows so that multiple messages are displayed at once.

2. The user can display the text of the message.

3. The user can display the mail header of the message and can choose to see all of the header lines or only a subset.

4. The user can suppress the display of header fields by name.

5. The user can save the text of the message, including the mail header lines that are chosen for display, as a text file, creating a new file or replacing an existing one.
6. The message attachments are displayed as icons. The icon selected for an attachment is based on the data typing information described in the **XCDI** specification, **Chapter 8**, **Data Typing**.
7. The user can save the data from an attachment as a file, creating a new file or replacing an existing one.
8. The user can access a menu with Actions for a selected attachment. Actions are invoked for an attachment by selecting an attachment and then selecting an action from the menu. If the ACTIONS attribute is defined for an attachment's data type, invoking the default action on an attachment activates the first Action listed in the Action attribute.

6.5.4 Composing New Messages

The following capabilities are supported for composing new messages:

1. The user can compose a message that includes both text and attachments and can specify and edit the To addressees, the Cc addressees, the Bcc (blind Cc) addressees and the Subject header field.
2. The user can compose multiple messages simultaneously.
3. The user can send messages to a file by including the file name as one of the To, Cc or Bcc addressees. The file name must be expressed as an absolute pathname or using a notation of ''+*file*,'' which selects the file named *file* in the default mailbox directory (see Section 6.5.1 on page 102).
4. The user can send messages to a mail alias, which represents one or more people or files, by including the alias name as one of the To, Cc or Bcc addressees.
5. The user can compose a new message starting with an empty body, Subject and addressees.
6. The user can select one or more messages from the mailbox message list and compose a new message that includes the text of these messages. The new message includes all of the attachments of the selected messages.
7. The user can select one or more messages from the mailbox message list and compose a forwarded message that includes the text and all the attachments from these messages. A forwarded message differs from a new message only in that the Subject header is pre-entered for the user, based on the Subject of one of the selected messages, and the text of the selected messages is identified as being forwarded.
8. The user can select one or more messages from the mailbox message list and compose a reply message that includes the text of these messages. If multiple messages are selected, multiple reply compositions are set up, so that each can be processed simultaneously and each has the correct addressees. The following variations on replies are supported:
    a. A Reply to Sender starts with an empty body, but pre-enters the To and Subject headers for the user, based on the sender and Subject of the original message. The reply contains no attachments unless the user takes explicit action to include them.
    b. A Reply to All is the same as a Reply to Sender, but all of the To and Cc addressees of the original message are pre-entered as To and Cc addressees, as appropriate, on the reply. The reply contains no attachments unless the user takes explicit action to include them.

c. A Reply to Sender with Include is the same as a Reply to Sender, but the message body starts with the text of the original message. The user can choose to indent each line of the text with a string of characters (such as ‘‘> ’’). The reply includes all attachments from the original message.

d. A Reply to All with Include is the same as a Reply to All, but the message body starts with the text of the original message. The user can choose to indent each line of the text with a string of characters (such as ‘‘> ’’). The reply includes all attachments from the original message.

9. The user can include a text file at the insertion point of the message being composed.

10. The user can write the header and text of the message being composed to a text file, creating a new file or replacing an existing one.

11. The user can manipulate the contents of the message text using the facilities described in Section 9.8 on page 194: word wrapping, finding and replacing text, and spell checking.

12. The user can select a mailbox into which all outgoing messages are saved.

13. The user can select a directory into which the mail services will attempt to save partial messages in case of interrupts or delivery errors.

14. The user can affect the attachments of mail as it is being composed:

    a. Add attachments

    b. Delete and undelete attachments

    c. Save the attachments as files

    d. Rename the attachments (which affects the name in the message, not the name of the file being attached)

6.5.5 Drag and Drop Capabilities

The XCDE mail services provide drag and drop capabilities as follows:

1. The user can drag the selected messages from the mailbox message list to any drop site registered to accept buffer drops. This achieves the following results:

    a. If dropped onto another mailbox message list, the messages are moved into the target mailbox.

    b. If dropped onto the text area of a message being composed, the texts of the messages are inserted into the new message.

    c. If dropped onto the attachments area of a message being composed, the messages are attached as a mailbox file.

    d. If dropped onto another application, the data should be treated as either a file or a collection of individual mail messages, as appropriate.

2. The user can drag text from the text area of a message being composed or viewed to any drop site registered to accept text drops. This achieves the following results:

    a. If dropped onto the text area of a message being composed, the text is inserted into the new message.

    b. If dropped onto the attachments area of a message being composed, the text is attached as a text attachment.

   c. If dropped onto another application, the data should be treated as text.

3. The user can drag an attachment from the attachment area of a message being composed or viewed to any drop site registered to accept buffer drops. This achieves the following results:

   a. If dropped onto the text area of a message being composed, text in the attachment is inserted into the new message.

   b. If dropped onto the attachments area of a message being composed, the attachment is copied into the attachments area of the new message.

   c. If dropped onto another application, the data should be treated in accordance with the type of data.

4. The user can drop files, text or data buffers into the attachment area of a message being composed, which causes new attachments to be added to the message.

6.5.6 Other Capabilities

The following other capabilities are supported:

1. The user can maintain (display, add entries, delete entries, change entries) a list of personal mail aliases.

2. The user can set up the mail services so that any incoming message is replied to automatically. The message is stored in the Inbox as usual. The user can specify the text and Subject of the reply message. The mail services may limit the number of these automatic replies sent to the same mail address.

Chapter 7

File Management Services

7.1 Introduction

The XCDE file management services provide the primary interface to the objects used in the X/Open Common Desktop Environment. These services provide a graphical interface for object and folder manipulation and for application execution.

The XCDE file management services interface consists of a window that shows the contents of a single folder or a set of nested folders. These services provide many manipulation functions, such as folder traversal, creating, moving, deleting objects and invoking actions (such as Edit or Print) on objects.

The XCDE file management services also provide windows for two special types of folders: application and trash. Application folders are used to organise application objects such as actions. The trash folder is a holding area for objects that a user wishes to delete.

7.2 Actions

This section defines the actions that provide XCDE file management services to support application portability at the C-language source or shell script levels.

NAME

dtfileaction — XCDE file management actions

SYNOPSIS

```
Dtfile [directory]
DtfileHome
Open directory
Open file
Print directory
Print file
```

DESCRIPTION

The XCDE File Management Services support the following file management actions:

Dtfile
Prompt the user for the pathname of a directory and open a folder view of the directory specified by the user.

Dtfile *directory*
Open a folder view of the directory named by the pathname in the *directory* argument.

DtfileHome
Open a folder view of the user's home directory.

Open *directory*
Open a folder view of the directory named by the pathname in the *directory* argument.

Open *file*
Open the file named by the pathname in the *file* argument. (The nature of the Open action is dependent on the type of file.)

Print *directory*
Print a listing of the directory named by the pathname in the *directory* argument.

Print *file*
Print the file named by the pathname in the *file* argument. (The nature of the Print action is dependent on the type of file.)

These actions can be invoked from an application using the *DtActionInvoke*( ) function or invoked from a command line using the *dtaction* utility.

SEE ALSO

dtaction, *DtActionInvoke*( ) in the **XCDI** specification.

CHANGE HISTORY

First released in Issue 1.

NAME

dttrashaction — XCDE trash management actions

SYNOPSIS

`Dttrash [`*`file`*`]`

DESCRIPTION

The XCDE File Management Services support the following trash management actions:

Dttrash

Open a folder view of the desktop trash folder.

Dttrash *file*

Move the file named by the pathname in the *file* argument to the desktop trash folder.

These actions can be invoked from an application using the *DtActionInvoke*( ) function or invoked from a command line using the *dtaction* utility.

SEE ALSO

dtaction, *DtActionInvoke*( ) in the **XCDI** specification.

CHANGE HISTORY

First released in Issue 1.

7.3 Messages

The XCDE file management services implement the *Display* and *Edit* requests for media type FILE_NAME. See the **XCDI** specification, **Section 6.6.2**, **Media Exchange Message Set**. These services also respond to the *Quit* desktop message. See the **XCDI** specification, **Section 6.6.1**, **Desktop Message Set**.

7.4 Capabilities

A conforming implementation of the XCDE file management services supports at least the following capabilities:

1. Provides file management services as described in the following subsections.
2. Conforms to the Required items in the Application Style Checklist in Chapter 20 on page 355, with the following exception that may exist on some implementations:
    a. There are certain operations that need not comply with checklist item 3-1. The selection of a folder in the iconic representation of a pathname and the ability to invoke the default action of the folder using the Enter key need not be available.
3. Has been internationalised using the standard interfaces in the X/Open CAE Specification, **System Interfaces and Headers**, **Issue 4, Version 2**, the X/Open CAE Specification, **Window Management: Xlib C Language Binding**, and the X/Open CAE Specification, **Motif Toolkit API**, and supports any locale available in the underlying X Window System Xlib implementation.

7.4.1 Folder Window

The XCDE file management services provide a window that shows the contents of a single folder or a set of nested folders. The user can configure how objects in a folder are displayed: as icons or textually. More than one folder window can be displayed at a time. The user can update folder windows and select/unselect individual or all objects in a folder. When a set of nested folders are displayed, the user can expand or collapse folder branches and control whether all objects in a folder or just the folder is shown.

The user can select from the following display modes:

- Objects are displayed in a sorted grid or can be placed by the user
- Objects can be displayed by name, by name with a large icon, by name with a small icon, or by name, date and size.
- Objects can be sorted alphabetically, by file type, by date or by size; objects sorted by file type are first sorted alphabetically by file type and are then sorted alphabetically by name within each file type
- Objects can be sorted in ascending or descending order
- Objects can be displayed based on the object type

7.4.2 Application Folder Window

The XCDE file management services provide a special window that shows the contents of an application folder. An application folder is a restricted folder that is used to organise application objects such as actions. The user is not allowed to traverse above the root application folder.

7.4.3 Trash Folder Window

The XCDE file management services provide a special window that shows the contents of the trash folder. The trash folder is a restricted folder that is used to store objects temporarily until the user asks to permanently remove them. The user is not allowed to traverse out of the trash folder. The user can restore objects from the trash folder if they have not been permanently removed.

7.4.4 Workspaces

The user can place frequently used objects on the workspace for easier accessibility. The user can also select actions for that object via a pop-up menu.

7.4.5 Object Movement and Modification

The XCDE file management services allow manipulation of objects in several different ways, based on appropriate permissions and the type of the object. The user can create, move, copy, create symbolic links to and delete applicable objects by using menu options and by using drag-and-drop. Objects can be renamed by using menu options or by mouse selection; the objects that cannot be renamed are implementation-dependent. Object permissions can be changed using menu options.

7.4.6 Object Search

The XCDE file management services allow searching for objects by name or by content. Searching by name performs a case-sensitive search for an object (or objects). Searching by content performs a case-insensitive search for a specified character string. By default, the current folder and its sub-folders are searched; the user can specify a different folder (and its sub-folders) to be searched. A list of objects matching the search criteria is shown to the user. For any object in the list, the user can choose to create a window to display the folder where the object is located or to place the object on the current workspace.

7.4.7 Folder Traversal

Simple folder traversal can be accomplished by invoking the default action on a folder icon. This action displays the contents of the selected folder. Traversal to a new folder by typing in the folder name and menu options for traversal to the user's home directory and parent folders are also supported.

7.4.8 Object Type/Action Association

An Actions menu is provided to show the Actions specified for each selected object. Actions are invoked for an object by selecting an object and then selecting an action from the Actions menu. If the ACTIONS attribute is defined for an object's data type, invoking the default action on an object activates the first Action listed in the Action attribute.

7.4.9 Registering Objects as Drop Sites

Every XCDE data type has three associated drop attributes: MOVE_TO_ACTION, COPY_TO_ACTION and LINK_TO_ACTION; see the **XCDI** specification, **Chapter 8**, **Data Typing**. The XCDE file management services register every object whose data type has a value for at least one of these attributes as a drop site. Objects can be dragged between different file management services folder windows, to workspaces and to cooperating clients. Direct manipulation can be used to supply objects as input to any user-defined action (for example, a Move action defined such that the dragged objects are moved to the dropped-on object).

7.4.10 Exit Services

Menu options are provided that allow the user to close file management services views.

The following information is saved as a consequence of exiting a desktop session:

1. The number and location of file management windows
2. The number and location of workspace objects
3. The number and location of objects placed on the workspace from file management services

Chapter 8

Front Panel Services

8.1 Introduction

The XCDE front panel services provide a key aspect of the XCDE user interface. The front panel appears in every workspace and provides access to the most commonly used facilities in the desktop. Considerable customisation is available to meet specific needs.

8.2 Formats

The front panel database provides definitions for the components that define the content and functionality of the front panel. Files containing front panel definitions must end with the **.fp** suffix. Like the action and data type database, the front panel database is constructed by reading all files ending in the **.fp** suffix found in the search path specified by the *DTDATABASESEARCHPATH* environment variable.

See the **XCDI** specification, **Section 8.4**, **Data Formats** for a complete description of the directory locations where these database files are found and for a description of the specific syntax for the database files.

8.2.1 File Format

The general syntax of the front panel configuration files is as follows:

```
set DtDbVersion=version_number
set VariableName=variable_value

RecordType record_name
{
        # Comment
        Keyword Value
        Keyword Value
                .
                .
                .
}
```

The *Comments*, *Versions* and *Variables* fields are described in the **XCDI** specification, **Section 8.4**, **Data Formats**.

The front panel record types each have a set of Keyword and Value pairs. There are six record types defined: PANEL, BOX, SUBPANEL, SWITCH, CONTROL and ANIMATION. Each record type has a set of keywords defined for it. Many of the keywords are used for multiple record types.

8.2.2 Record Types

PANEL *front panel name*

The PANEL record type defines the outermost container of the front panel. It can contain one or more BOXes and optionally repositioning handles, a menu and a minimise button. The keywords defined for PANEL are described in the following table.

PANEL Record Type Keywords and Values		
Keyword	**Value**	**Default**
CONTROL_BEHAVIOR	(**double_click** / **single_click**)	single_click
DISPLAY_CONTROL_LABELS	(**True** / **False**)	False
DISPLAY_HANDLES	(**True** / **False**)	True
DISPLAY_MENU	(**True** / **False**)	True
DISPLAY_MINIMIZE	(**True** / **False**)	True
HELP_STRING	*string*	NULL
HELP_TOPIC	*topic name*	NULL
HELP_VOLUME	*volume name*	FPanel
LOCKED	(**True** / **False**)	False
PANEL_GEOMETRY	{+−}*xoffset*[{+−}*yoffset*]	NULL
RESOLUTION	(**high** / **medium** / **low** / **match_display**)	match_display
SUBPANEL_UNPOST	(**True** / **False**)	True

BOX *box name*

The BOX record type defines a container within a PANEL that can hold a row of CONTROLS and at most one SWITCH container. Multiple BOXes within a PANEL are stacked vertically. The keywords defined for BOX are described in the following table.

BOX Record Type Keywords and Values		
Keyword	**Value**	**Default**
CONTAINER_NAME	front panel name	NULL (required)
DELETE	(**True** / **False**)	False
HELP_STRING	*string*	NULL
HELP_TOPIC	*topic name*	NULL
HELP_VOLUME	*volume name*	FPanel
LOCKED	(**True** / **False**)	False
POSITION_HINTS	(**first** / **last** / *integer* ≥ 1)	first

SUBPANEL *subpanel name*

The SUBPANEL record type defines a secondary container for CONTROLs that slide up from the front panel. SUBPANELs can also contain a drop zone where new controls can be dynamically added. The keywords defined for SUBPANEL are described in the following table.

SUBPANEL Record Type Keywords and Values		
Keyword	**Value**	**Default**
CONTAINER_NAME	*control name*	NULL (required)
CONTROL_INSTALL	(**True** / **False**)	True
DELETE	(**True** / **False**)	False
HELP_STRING	*string*	NULL
HELP_TOPIC	*topic name*	NULL
HELP_VOLUME	*volume name*	FPanel
LOCKED	(**True** / **False**)	False
TITLE	*string*	NULL

SWITCH *switch name*

The SWITCH record type defines a container within a BOX that contains a set of push buttons, each of which give access to a corresponding workspace. See *dtwm* for a description of the multiple workspace capabilities. Also contained within the SWITCH container is an optional set of CONTROLs. These are presented in a column on each side of the push buttons. The keywords defined for SWITCH are described in the following table.

SWITCH Record Type Keywords and Values		
Keyword	**Value**	**Default**
CONTAINER_NAME	*box name*	NULL (required)
DELETE	(**True** / **False**)	False
HELP_STRING	*string*	NULL
HELP_TOPIC	*topic name*	NULL
HELP_VOLUME	*volume name*	FPanel
LOCKED	(**True** / **False**)	False
NUMBER_OF_ROWS	*integer*	2
POSITION_HINTS	(**first** / **last** / *integer* ≥ 1)	first

CONTROL *control name*

The CONTROL record type defines the main functional component of the front panel. CONTROLs typically have actions defined for them that are invoked on selection or drag and drop or both. CONTROLs are displayed with icons or labels or both and can have iconic animations associated with them. The keywords defined for CONTROL are described in the following table.

CONTROL Record Type Keywords and Values		
Keyword	**Value**	**Default**
ALTERNATE_ICON	*image name*	NULL
CLIENT_GEOMETRY	*width* **x** *height*	NULL
CLIENT_NAME	*client name*	NULL
CONTAINER_NAME	(*box name*/*switch name*/*subpanel name*)	NULL (required)
CONTAINER_TYPE	(**BOX**/**SWITCH**/**SUBPANEL**)	NULL (required)
DATE_FORMAT	*format string*	%b%n%e
DELETE	(**True**/**False**)	False
DROP_ACTION	*action_name*	NULL
DROP_ANIMATION	*animation name*	NULL
FILE_NAME	*pathname*	NULL
HELP_STRING	*string*	NULL
HELP_TOPIC	*topic name*	NULL
HELP_VOLUME	*volume name*	FPanel
ICON	*image name*	NULL
LABEL	*string*	NULL
LOCKED	(**True**/**False**)	False
MONITOR_TYPE	(**none**/**mail**/**file**)	none
POSITION_HINTS	(**first**/**last**/*integer* ≥ 1)	first
PUSH_ACTION	*action_name*	NULL
PUSH_ANIMATION	*animation name*	NULL
PUSH_RECALL	(**True**/**False**)	False
TYPE	(**blank**/**busy**/**client**/**clock**/**date**/**file**/**icon**)	icon

ANIMATION *animation name*

The ANIMATION record types are a sequence of image name and time delay pairs that are displayed by a CONTROL on a PUSH_ACTION or DROP_ACTION.

ANIMATION Record Type Keywords and Values		
Keyword	**Value**	**Default**
ANIMATION	*image name* [*millisecond delay*]	None [200]

8.2.3 Keyword and Value Descriptions

The following list contains a description of each of the keywords defined by the front panel.

ALTERNATE_ICON

Used with record types of: CONTROL.

Used with control types of: **busy** and **icon**.

ALTERNATE_ICON defines an image to be used to replace the normal image within a control with a mail or file value for the MONITOR_TYPE. The image is displayed when the file being monitored changes. For the **busy** control, ALTERNATE_ICON is cycled with ICON to give the blinking effect.

ANIMATION

Used with record types of: ANIMATION.

Used with control keywords of: PUSH_ANIMATION and DROP_ANIMATION.

ANIMATION defines a sequence of images to be displayed for either a PUSH_ACTION or DROP_ACTION. Each animation within the list is displayed in order with a default time separation of 200 milliseconds. For a slower or faster sequence, the image name value can be followed by the amount of time to display the image. If no time value is specified, the previously specified value is used.

CLIENT_GEOMETRY

Used with record types of: CONTROL.

Used with control type of: **client**.

Used with control keywords of: CLIENT_NAME.

CLIENT_GEOMETRY specifies the size (in pixels) needed for the window of a client displayed within the front panel.

CLIENT_NAME

Used with record types of: CONTROL.

Used with control types of: **client** or **icon**.

Used with control keywords of: PUSH_RECALL.

CLIENT_NAME specifies a name used to associate a control with an executable. It is necessary for control types of **client** (an X client running within the front panel) and for **icon** when the keyword PUSH_RECALL is True. The value *client name* is the name of the executable or can be set via a command-line argument for some clients (such as *xterm* –**name** *panelterm*). The first string of the WM_CLASS property is the value used.

CONTAINER_NAME

Used with record types of: BOX, SUBPANEL, SWITCH and CONTROL.

CONTAINER_NAME associates a component with its parent. For example, the CONTAINER_NAME value for a SWITCH tells the front panel into which BOX it should be placed. Since controls can reside in several different component types, CONTAINER_NAME is used in conjunction with CONTAINER_TYPE to define a control's parent.

CONTAINER_TYPE

Used with record types of: CONTROL.

CONTAINER_TYPE defines a control's parent type. This is used to identify a control uniquely so that it can be created within the proper parent.

CONTROL_BEHAVIOR

Used with record types of: PANEL.

CONTROL_BEHAVIOR provides the mechanism for setting the user model for front panel controls. Controls can be set to invoke their PUSH_ACTION by either a single or double click.

CONTROL_INSTALL

Used with record types of: SUBPANEL.

CONTROL_INSTALL enables or disables dynamic control installation into subpanels. A value of True causes the control installation area to be displayed within the subpanel.

DATE_FORMAT

Used with record types of: CONTROL.

Used with control types of: **date**.

DATE_FORMAT specifies the layout of the date string for a control of type **date**. The format is the same used by the X/Open CAE Specification, **System Interfaces and Headers**, **Issue 4, Version 2** *strftime*( ) function.

DELETE

Used with record types of: BOX, SUBPANEL, SWITCH and CONTROL.

DELETE is used to override and remove a non-locked component from the front panel. This is necessary to eliminate system default front panel components without replacing the default files. To use DELETE, a copy of the component definition with the additional DELETE keyword is stored in a file with an **.fp** suffix in the user's or application type's directory.

DISPLAY_CONTROL_LABELS

Used with record types of: PANEL.

DISPLAY_CONTROL_LABELS specifies whether the controls contained within the front panel's boxes have their labels displayed.

DISPLAY_HANDLES

Used with record types of: PANEL.

If DISPLAY_HANDLES is set to True, the move handles are displayed along the left and right edges of the front panel.

DISPLAY_MENU

Used with record types of: PANEL.

If DISPLAY_MENU is set to True, and DISPLAY_HANDLES is also set to True, the system menu button is displayed in the upper left hand corner of the front panel.

DISPLAY_MINIMIZE

Used with record types of: PANEL.

If DISPLAY_MINIMIZE is set to True, and DISPLAY_HANDLES is also set to True, the minimise button is displayed in the upper right hand corner of the front panel.

DROP_ACTION

Used with record types of: CONTROL.

DROP_ACTION specifies the function that is invoked when a drop occurs on the control.

DROP_ANIMATION

Used with record types of: CONTROL.

Used with all control types except: **client**.

DROP_ANIMATION specifies the name of an animation record to be displayed when a drop occurs on the control. The control must have a DROP_ACTION defined for the animation to be used.

FILE_NAME

Used with record types of: CONTROL.

Used with control types of: **icon** and **file**.

For the control type of **file**, FILE_NAME is the pathname of the file to be used for the control. The file's file type, actions and images are extracted from the action database for the control.

For MONITOR_TYPE **mail** and **file**, FILE_NAME specifies the pathname of the file the front panel monitors for either mail arrival or some other user-defined data.

For MONITOR_TYPE **file**, the ALTERNATE_ICON image is displayed when the monitored file is non-empty. The ICON image is displayed if the file is empty or non-existent.

For MONITOR_TYPE **mail**, the ALTERNATE_ICON image is displayed when the file increases in size.

HELP_STRING

Used with record types of: PANEL, BOX, SUBPANEL, SWITCH and CONTROL.

HELP_STRING specifies an on-line help string to display when help is requested over a front panel component. Since each component type can have a help string associated with it, there is a precedence used in deciding which help string to display. Control help takes precedence over box help, switch help and subpanel help. Box help takes precedence over subpanel help. The HELP_STRING value is used only if no HELP_TOPIC value is defined.

HELP_TOPIC

Used with record types of: PANEL, BOX, SUBPANEL, SWITCH and CONTROL.

HELP_TOPIC specifies an on-line help topic that is used with either the default help volume or a help volume specified by the HELP_VOLUME keyword to display help information when requested over a front panel component. Like the HELP_STRING keyword, each component type can have a help topic associated with it and the same precedence rules are used.

HELP_VOLUME

Used with record types of: PANEL, BOX, SUBPANEL, SWITCH and CONTROL.

When HELP_VOLUME is used in conjunction with HELP_TOPIC, it defines the help information to be displayed for a front panel component.

ICON

Used with record types of: CONTROL.

Used with control types of: **icon**, **file** and **busy**.

ICON specifies the image to be displayed within a control.

LABEL

Used with record types of: CONTROL.

Used with control types of: **icon**, **file**, **clock** and **busy**.

LABEL specifies the string to be displayed when a control is in a subpanel.

LOCKED

Used with record types of: PANEL, BOX, SUBPANEL, SWITCH and CONTROL.

LOCKED provides the mechanism to prevent a component definition of identical type, name and parent from overriding (replacing) this definition. Since the front panel can be defined within multiple configuration files, it provides the flexibility to override components found earlier in the search path. Like actions and file types, this allows the front panel to be customised at several levels.

MONITOR_TYPE

Used with record types of: CONTROL.

Used with control types of: **icon** and **file**

MONITOR_TYPE specifies the method of checking the file being monitored, specified by the keyword FILE_NAME.

NUMBER_OF_ROWS

Used with record types of: SWITCH.

NUMBER_OF_ROWS provides control over the layout of the workspace switch buttons. The switch buttons are arranged in a row and column layout with the NUMBER_OF_ROWS keyword defining the number of rows in the layout. The number of columns is derived from this value and the total number of switch buttons.

PANEL_GEOMETRY

Used with record types of: PANEL.

PANEL_GEOMETRY specifies a non-default location to position the front panel when it is created. By default, the front panel is centered along the bottom of the display.

POSITION_HINTS

Used with record types of: BOX, SWITCH and CONTROL.

POSITION_HINTS specifies the ordering of boxes in the front panel, the switch and controls in boxes, and controls in subpanels. When two components have the same value for POSITION_HINTS, the first one read from the configuration file is placed first.

PUSH_ACTION

Used with record types of: CONTROL.

PUSH_ACTION specifies the function that is invoked when a selection occurs on the control.

PUSH_ANIMATION

Used with record types of: CONTROL.

Used with all control types except: **client**.

PUSH_ANIMATION specifies the name of an animation record to be displayed when a selection occurs on the control. The control must have a PUSH_ACTION defined for the animation to be used.

PUSH_RECALL

Used with record types of: CONTROL.

Used with control keywords of: CLIENT_NAME.

When PUSH_RECALL is set to True, it specifies that only one process can be started by the control. If the process is already running, it is displayed within the current workspace and shuffled to the top of the window stack. The value for the CLIENT_NAME keyword is used to identify the process for push recall behaviour.

RESOLUTION

Used with record types of: PANEL.

RESOLUTION allows the icon set for the front panel to be forced to a particular set. By default, the front panel determines the display resolution at runtime and chooses the high resolution icon set if the display width is 1024 pixels or wider.

SUBPANEL_UNPOST

Used with record types of: PANEL.

When SUBPANEL_UNPOST is set to True, it causes a subpanel to hide itself whenever a PUSH_ACTION occurs on one of the subpanel controls. If the subpanel has been torn off of the front panel, the behaviour of the subpanel is forced to remain posted on PUSH_ACTION.

TITLE

Used with record types of: SUBPANEL.

TITLE specifies the string to be displayed in the title area of the subpanel.

TYPE

Used with record types of: CONTROL.

A number of different control types are defined by the front panel:

blank	Space-holder control
busy	Busy light
client	A client window
clock	Front panel clock
date	Front panel date
file	References a file on the file system and uses that file's actions and image
icon	Front panel general control

8.3 Capabilities

A conforming implementation of the XCDE front panel services supports at least the following capabilities:

1. Provides front panel services as described in the following subsections.
2. Conforms to the Required items in the Application Style Checklist in Chapter 20 on page 355.
3. Has been internationalised using the standard interfaces in the X/Open CAE Specification, **System Interfaces and Headers**, **Issue 4, Version 2**, the X/Open CAE Specification, **Window Management: Xlib C Language Binding**, and the X/Open CAE Specification, **Motif Toolkit API**, and supports any locale available in the underlying X Window System Xlib implementation.

8.3.1 General Layout

The default configuration of the front panel is as follows:

1. The front panel appears in every workspace.
2. The front panel includes controls for the most commonly used actions, as listed in the next section. Selecting a control invokes the action associated with the control.
3. Controls in the front panel may have indicators associated with them whose selection causes a subpanel to appear.
4. Subpanels contain controls that provide access to additional actions.
5. Subpanels may have a special control that adds an icon to the subpanel. The user can drop an icon on the special control, which causes that icon to become a control in the subpanel.
6. The user can move and iconify the front panel.

8.3.2 Special Controls

The front panel services provide the following special controls by default:

1. The file manager control depicts a file folder. When it is selected, it invokes the **DtfileHome** action (see Section 7.2 on page 107), which opens a view of the user's home directory.
2. The lock control depicts a lock. When it is selected, it invokes the **LockDisplay** action (see Section 3.4 on page 30), which prevents access to the system until the user's password is entered.
3. The exit control, when it is selected, invokes the **ExitSession** action (see Section 3.4 on page 30), which logs the user out of the current desktop session.
4. The printer control prints files that are dropped on it, by invoking the **Print** action (see Section 17.2 on page 333). At least text files are supported. Its icon shows a printer.
5. The application manager control, when it is selected, invokes the **Dtappmgr** action (see Section 15.3 on page 327), which shows applications installed on the system.
6. The help control, when it is selected, invokes the **Dthelpview** action (see Section 4.5 on page 62), which displays the top-level help topic for the desktop.
7. The workspace controls move users among available workspaces. There is one control for each workspace. Selecting a workspace control moves the user into that workspace.
8. The launch light indicates that an action is in the process of starting.
9. The trash can control removes files that are dropped on it by invoking the **Dttrash** action (see Section 7.2 on page 107).

8.3.3 Other Capabilities

The following additional capabilities are provided:

1. Animation for drag and drop actions.
2. The user can add and delete workspaces.
3. The user can provide custom names for each workspace.
4. The user can add and remove subpanels.

Chapter 9

Text Editing Services

9.1 Introduction

The XCDE text editing services provide a tablet to create and edit short documents such as memos and mail messages in the character set of the current locale.

9.2 Widgets

This section defines the widget classes that provide XCDE text editing services to support application portability at the C-language source level.

NAME

DtEditor — the DtEditor widget class

SYNOPSIS

```
#include <Dt/Editor.h>
```

DESCRIPTION

The DtEditor widget supports creating and editing text files. It gives applications running in the desktop environment a consistent method for editing text data. The widget consists of:

- A scrolled edit window for text
- Dialogs for finding and changing text
- Formatting options
- Convenience functions for programmatically controlling the widget

The DtEditor widget supports the following set of basic editing operations:

- Finding and changing text
- Simple formatting
- Undoing the previous edit operation

All operations support locales with single- and multi-byte characters.

The DtEditor widget also supports multi-byte text and buffers of data. Data can be passed between the application and the DtEditor widget, or a file and the widget.

The DtEditor widget provides separate callback lists to track when text is selected or deselected. In addition, it extends the standard help callback to report help requests from any of its components.

Widget subclassing is not supported for the DtEditor widget class.

Edit Window

The edit window supports basic editing operations such as cut and paste, find and change, and simple formatting.

Mouse and Keyboard

The user can use the mouse to move the edit cursor and to select portions of a document for editing operations. Selection is based on the model specified in the *Inter-Client Communication Conventions Manual* (ICCCM—see the X/Open CAE Specification, **Window Management: File Formats and Application Conventions**). The DtEditor widget supports primary and secondary selection.

The user can cut, copy and paste text using the clipboard, primary transfer or secondary transfer. The DtEditor widget accepts drops of text, text files or buffers of data. Text drops are inserted where the mouse button is released to complete the drop. Dropped files and buffers of data are placed at the insertion cursor. The DtEditor widget supports dragging of text within the edit window or to a different widget.

The DtEditor widget provides a set of translations for the edit window. The default translations provide key bindings for moving the insertion cursor, and deleting, inserting and selecting text. The insertion cursor, displayed as an I-beam, shows where input is inserted. Input is inserted just before the insertion cursor.

Dialogs

The DtEditor widget includes dialogs to provide a graphical user interface to its functionality:

- Find / Change dialog
- Format Settings dialog

The titles of all dialogs are controlled with the **DtNdialogTitle** resource. All dialogs are posted using corresponding convenience functions and remain posted until dismissed by the user. Each dialog includes Close and Help buttons in addition to buttons described in the following lists.

The Find / Change dialog for the DtEditor widget enables users to search for, and optionally replace, a string in the edit window. The dialog includes fields for specifying the find string and the replacement string. When the user initiates a Find, the next occurrence of the specified string (regular expressions are not supported) is highlighted in the DtEditor widget, if found; otherwise, the DtEditor widget displays a message dialog stating the string was not found. If the string was found, the user has the option to change the highlighted occurrence or all occurrences.

The *DtEditorFind*( ) and *DtEditorChange*( ) functions provide a programmatic interface to the find and change functionality of the DtEditor widget.

The Format Settings dialog for the DtEditor widget enables users to format the contents of the edit window, format just the paragraph containing the insertion cursor, or specify the arguments used when formatting text. The arguments include margin settings and text alignment. The user has the choice of aligning the text flush with the left or right margin, centering each line of text between the margins, or aligning it flush with both margins.

The *DtEditorFormat*( ) function provides a programmatic interface to the format functionality of the DtEditor widget.

Word Wrap and Formatting

Word wrap and text formatting are essentially independent operations. Word wrap pertains to the dynamic display of lines, as delimited by <newline> characters, which exceed the width of the Text Editor window and is based on the left and right window boundaries. When word wrap mode is off (the default), each line of text is displayed on a single line on the display and text entered at the right window boundary causes the window to scroll automatically to the right to accommodate the new text until an actual <newline> character is entered (normally, by pressing the Return key). When word wrap mode is on, lines longer than the window width are automatically wrapped at the right window margin to one or more display lines, and text entered at the right window boundary is automatically broken on a word boundary to the first column of the next display line. Word wrap is dynamic in that word-wrapped lines are automatically adjusted when text is inserted or deleted or when the window is resized. Word wrap only affects the display of lines; it does not actually insert <newline> characters in the text.

Text formatting is a static operation that inserts actual <newline> (and / or <space>) characters directly in the text to match it to the left and right margins (and justification mode) specified in the Format Settings dialog. Format settings affect text only when explicitly applied and have no affect on word wrap or previously formatted text. Initially, and whenever the window is resized, the right format margin is automatically set to the window width to match the word wrap boundary.

Classes

The DtEditor widget inherits behaviour and resources from *Core*, *Composite*, *Constraint*, *XmManager*, *XmBulletinBoard* and *XmForm* classes.

The class pointer is **dtEditorWidgetClass**.

The class name is *DtEditorWidget*.

New Resources

The following table defines a set of widget resources the application uses to specify data. The application can also set the resource values for the inherited classes to set attributes for this widget. To reference a resource by name or by class in a **.Xdefaults** file, the application must remove the **DtN** or **DtC** prefix and use the remaining letters. To specify one of the defined values for a resource in a **.Xdefaults** file, the application must remove the **Dt** prefix and use the remaining letters (in either lower case or upper case, but including any underscores between words). The codes in the access column indicate if the given resource can be set at creation time (C), set by using *XtSetValues*( ) (S), retrieved by using *XtGetValues*( ) (G), or is not applicable (N/A).

DtEditor Resource Set				
Name	**Class**	**Type**	**Default**	**Access**
DtNautoShowCursorPosition	**DtCAutoShowCursorPosition**	**Boolean**	True	CSG
DtNblinkRate	**DtCBlinkRate**	**int**	500	CSG
DtNbuttonFontList	**DtCFontList**	**XmFontList**	dynamic	CSG
DtNcolumns	**DtCColumns**	**XmNcolumns**	dynamic	CSG
DtNcursorPosition	**DtCCursorPosition**	**XmTextPosition**	0	CSG
DtNcursorPositionVisible	**DtCCursorPositionVisible**	**Boolean**	True	CSG
DtNdialogTitle	**DtCDialogTitle**	**XmString**	NULL	CSG
DtNeditable	**DtCEditable**	**Boolean**	True	CSG
DtNlabelFontList	**DtCFontList**	**XmFontList**	dynamic	CSG
DtNmaxLength	**DtCMaxLength**	**int**	largest integer	CSG
DtNoverstrike	**DtCOverstrike**	**Boolean**	False	CSG
DtNrows	**DtCRows**	**XmNrows**	dynamic	CSG
DtNscrollHorizontal	**DtCScroll**	**Boolean**	True	CG
DtNscrollLeftSide	**DtCScrollSide**	**Boolean**	dynamic	CG
DtNscrollTopSide	**DtCScrollSide**	**Boolean**	False	CG
DtNscrollVertical	**DtCScroll**	**Boolean**	True	CG
DtNtextBackground	**DtCBackground**	**Pixel**	dynamic	CSG
DtNtextDeselectCallback	**DtCCallback**	**XtCallbackList**	NULL	C
DtNtextFontList	**DtCFontList**	**XmFontList**	dynamic	CSG
DtNtextForeground	**DtCForeground**	**Pixel**	dynamic	CSG
DtNtextSelectCallback	**DtCCallback**	**XtCallbackList**	NULL	C
DtNtextTranslations	**DtCTranslations**	**XtTranslations**	NULL	CS
DtNtopCharacter	**DtCTextPosition**	**XmTextPosition**	0	CSG
DtNwordWrap	**DtCWordWrap**	**Boolean**	False	CSG

DtNautoShowCursorPosition

Ensures that the text visible in the scrolled edit window contains the insert cursor when set to True. If the insert cursor changes, the contents of the DtEditor widget may scroll in order to bring the insertion point into the window.

DtNblinkRate

Specifies the blink rate of the text cursor in milliseconds. The time indicated in the blink rate relates to the time the cursor is visible and the time the cursor is invisible (that is, the time it takes to blink the insertion cursor on and off is twice the blink

rate). The cursor does not blink when the blink rate is set to zero. The value cannot be negative.

DtNbuttonFontList
Specifies the font list used for the DtEditor buttons (the buttons appearing in the DtEditor dialogs). If this value is NULL at initialisation, it is initialised by looking up the parent hierarchy of the widget for an ancestor that is a subclass of the *XmBulletinBoard*, *VendorShell* or *XmMenuShell* widget class. If such an ancestor is found, the font list is initialised to the appropriate default font list of the ancestor widget (**XmNdefaultFontList** for *VendorShell* and *XmMenuShell*, and **XmNbuttonFontList** for *XmBulletinBoard*). If no such ancestor is found, the default is implementation dependent.

DtNcolumns
Specifies the initial width of the edit window of the DtEditor widget as an integral number of characters. The width equals the number of characters this resource specifies multiplied by the maximum character width of the associated font. For proportionate fonts, the actual number of characters that fit on a given line may be greater than the value specified. The value must be greater than zero. The default value depends on the value of the **DtNwidth** resource.

DtNcursorPosition
Indicates the position in the DtEditor widget where the current insert cursor is located. This position is determined by the number of characters from the beginning of the text. The first character position is zero.

DtNcursorPositionVisible
When set to True, this resource specifies that the insert cursor position is marked by a blinking text cursor.

DtNdialogTitle
Specifies an XmString that appears as part of the titles for the dialogs displayed by the DtEditor widget. If this resource is non-NULL, it is used as the prefix of the titles for the Find / Change and the Format Settings dialogs.

DtNeditable
When set to True, this resource indicates that the user can edit the text; otherwise, it prohibits the user from editing the text.

DtNlabelFontList
Specifies the font list used for the labels for DtEditor (the labels appear in the DtEditor dialogs). If this value is NULL at initialisation, it is initialised by looking up the parent hierarchy of the widget for an ancestor that is a subclass of the *XmBulletinBoard*, *VendorShell* or *XmMenuShell* widget class. If such an ancestor is found, the font list is initialised to the **XmNlabelFontList** of the ancestor widget. If no such ancestor is found, the default is implementation dependent.

DtNmaxLength
Specifies the maximum length of the text string that can be entered into the DtEditor widget from the keyboard. This value must be non-negative. Strings entered using the *DtEditorSetContents*( ) or *DtEditorSetContentsFromFile*( ) functions ignore this resource.

DtNoverstrike
When set to False, characters typed into the DtEditor widget are inserted at the position of the insertion cursor. This is the default behaviour. If set to True, characters typed into the DtEditor widget replace the characters that directly follow

the insertion cursor. When the end of the line is reached, characters are appended to the end of the line.

DtNrows
Specifies the initial height of the edit window of the DtEditor widget measured in character heights. The value must be greater than zero.

DtNscrollHorizontal
When set to True, this resource adds a ScrollBar that allows the user to scroll horizontally through text.

DtNscrollLeftSide
When set to True, this resource indicates that the vertical ScrollBar should be placed on the left side of the scrolled edit window. This attribute is ignored if **DtNscrollVertical** is False. The default value may depend on the value of the **XmNstringDirection** resource.

DtNscrollTopSide
When set to True, this resource indicates that the horizontal ScrollBar should be placed on the top side of the scrolled edit window. This attribute is ignored if **DtNscrollHorizontal** is False.

DtNscrollVertical
When set to True, this resource adds a ScrollBar that allows the user to scroll vertically through text.

DtNtextBackground
Specifies the background of the edit window and the text fields for DtEditor (the text fields appear in the DtEditor dialogs).

DtNtextDeselectCallback
Specifies a function called whenever the selection becomes NULL (that is, no text is selected within the edit area). The reason sent by the callback is DtEDITOR_TEXT_DESELECT.

DtNtextFontList
Specifies the font list used for the edit window and the text fields for DtEditor (the text fields appear in the DtEditor dialogs). If this value is NULL at initialisation, it is initialised by looking up the parent hierarchy of the widget for an ancestor that is a subclass of the *XmBulletinBoard* or *VendorShell* widget class. If such an ancestor is found, the font list is initialised to the **XmNtextFontList** of the ancestor widget. If no such ancestor is found, the default is implementation dependent.

DtNtextForeground
Specifies the foreground of the edit window and the text fields for DtEditor (the text fields appear in the DtEditor dialogs).

DtNtextSelectCallback
Specifies a function called whenever the selection becomes non-NULL (that is, some text is selected within the edit area). The reason sent by the callback is DtEDITOR_TEXT_SELECT.

DtNtextTranslations
Adds translations to the edit window. Translations specified with **DtNtextTranslations** override any duplicate translations defined for the edit window.

DtNtopCharacter

Displays the line that contains the position of text at the top of the scrolled edit window. The line is displayed at the top of the DtEditor widget without shifting the text left or right. The position is determined by the number of characters from the beginning of the text. The first character position is zero.

XtGetValues( ) for **DtNtopCharacter** returns the position of the first character in the line that is displayed at the top of the DtEditor widget.

DtNwordWrap

Indicates that text not go off the right edge of the window, but that lines are broken at word breaks with soft line feeds when they reach the right edge of the window.

Word wrap affects only the visual appearance of the contents of a DtEditor widget. The line breaks (soft line feeds) are not physically inserted into the text. The DtEditor widget supports substituting <newline>s when the contents of the widget are retrieved or saved to a file (see *DtEditorGetContents*( ) and *DtEditorSaveContentsToFile*( )).

Localisation Resources

The following table defines a set of widget resources designed for localisation of the DtEditor widget and its dialogs. Default values for these resources depends on the locale.

DtEditor Localisation Resource Set				
Name	**Class**	**Type**	**Default**	**Access**
DtNfindChangeDialogTitle	**DtCFindChangeDialogTitle**	**XmString**	Dynamic	CSG
DtNformatSettingsDialogTitle	**DtCFormatSettingsDialogTitle**	**XmString**	Dynamic	CSG
DtNinformationDialogTitle	**DtCInformationDialogTitle**	**XmString**	Dynamic	CSG

DtNfindChangeDialogTitle

Specifies the title for the Find/Change dialog. If **DtNdialogTitle** is non-NULL, it is added as a prefix to this resource to form the title. The default value in the C locale is **Find/Change**.

DtNformatSettingsDialogTitle

Specifies the title for the Format Settings dialog. If **DtNdialogTitle** is non-NULL, it is added as a prefix to this resource to form the title. The default value in the C locale is **Format Settings**.

DtNinformationDialogTitle

Specifies the title for the Information dialog used to present feedback and general information to the user. If **DtNdialogTitle** is non-NULL, it is added as a prefix to this resource to form the title. The default value in the C locale is **Information**.

Inherited Resources

The DtEditor widget inherits behaviour and resources from the following named superclasses. For a complete description of each resource, see the entry in X/Open CAE Specification, **Motif Toolkit API** for that superclass.

XmForm Resource Set				
Name	**Class**	**Type**	**Default**	**Access**
XmNfractionBase	**XmCMaxValue**	**int**	100	CSG
XmNhorizontalSpacing	**XmCSpacing**	**Dimension**	0	CSG
XmNrubberPositioning	**XmCRubberPositioning**	**Boolean**	False	CSG
XmNverticalSpacing	**XmCSpacing**	**Dimension**	0	CSG

XmBulletinBoard Resource Set				
Name	**Class**	**Type**	**Default**	**Access**
XmNallowOverlap	**XmCAllowOverlap**	**Boolean**	True	CSG
XmNautoUnmanage	**XmCAutoUnmanage**	**Boolean**	True	CG
XmNbuttonFontList	**XmCButtonFontList**	**XmFontList**	dynamic	CSG
XmNcancelButton	**XmCWidget**	**Window**	NULL	SG
XmNdefaultButton	**XmCWidget**	**Window**	SG	
XmNdefaultPosition	**XmCDefaultPosition**	**Boolean**	True	CSG
XmNdialogStyle	**XmCDialogStyle**	**unsigned char**	dynamic	CSG
XmNdialogTitle	**XmCDialogTitle**	**XmString**	NULL	CSG
XmNfocusCallback	**XmCCallback**	**XtCallbackList**	NULL	C
XmNlabelFontList	**XmCLabelFontListk**	**XmFontList**	dynamic	CSG
XmNmapCallback	**XmCCallback**	**XtCallbackList**	NULL	C
XmNmarginHeight	**XmCMarginHeight**	**Dimension**	10	CSG
XmNmarginWidth	**XmCMarginWidth**	**Dimension**	10	CSG
XmNnoResize	**XmCNoResize**	**Boolean**	False	CSG
XmNresizePolicy	**XmCResizePolicy**	**unsigned char**	XmRESIZE_ANY	CSG
XmNshadowType	**XmCShadowType**	**unsigned char**	XmSHADOW_OUT	CSG
XmNtextFontList	**XmCTextFontList**	**XmFontList**	dynamic	CSG
XmNtextTranslations	**XmCTranslations**	**XtTranslations**	NULL	C
XmNunmapCallback	**XmCCallback**	**XtCallbackList**	NULL	C

XmManager Resource Set				
Name	**Class**	**Type**	**Default**	**Access**
XmNbottomShadowColor	**XmCBottomShadowColor**	**Pixel**	dynamic	CSG
XmNbottomShadow-Pixmap	**XmCBottomShadow-Pixmap**	**Pixmap**	XmUNSPECIFIED-_PIXMAP	CSG
XmNforeground	**XmCForeground**	**Pixel**	dynamic	CSG
XmNhelpCallback	**XmCCallback**	**XtCallbackList**	NULL	C
XmNhighlightColor	**XmCHighlightColor**	**Pixel**	dynamic	CSG
XmNhighlightPixmap	**XmCHighlightPixmap**	**Pixmap**	dynamic	CSG
XmNinitialFocus	**XmCInitialFocus**	**Widget**	NULL	CSG
XmNnavigationType	**XmCNavigationType**	**XmNavigation-Type**	dynamic	CSG
XmNshadowThickness	**XmCShadowThickness**	**Dimension**	dynamic	CSG
XmNstringDirection	**XmCStringDirection**	**XmString-Dynamic**	dynamic	CG
XmNtopShadowColor	**XmCTopShadowColor**	**Pixel**	dynamic	CSG
XmNtopShadowPixmap	**XmCTopShadowPixmap**	**Pixmap**	dynamic	CSG
XmNtraversalOn	**XmCTraversalOn**	**Boolean**	dynamic	CSG
XmNunitType	**XmCUnitType**	**unsigned char**	dynamic	CSG
XmNuserData	**XmCUserData**	**XtPointer**	NULL	CSG

Composite Resource Set				
Name	**Class**	**Type**	**Default**	**Access**
XmNchildren	**XmCReadOnly**	**WidgetList**	NULL	G
XmNinsertPosition	**XmCInsertPosition**	**XtOrderProc**	default procedure	CSG
XmNnumChildren	**XmCReadOnly**	**Cardinal**	0	G

Core Resource Set				
Name	**Class**	**Type**	**Default**	**Access**
XmNaccelerators	**XmCAccelerators**	**XtAccelerators**	dynamic	CSG
XmNancestorSensitive	**XmCSensitive**	**Boolean**	dynamic	G
XmNbackground	**XmCBackground**	**Pixel**	dynamic	CSG
XmNbackgroundPixmap	**XmCPixmap**	**Pixmap**	XmUNSPECIFIED-_PIXMAP	CSG
XmNborderColor	**XmCBorderColor**	**Pixel**	XtDefaultForeground	CSG
XmNborderPixmap	**XmCPixmap**	**Pixmap**	XmUNSPECIFIED-_PIXMAP	CSG
XmNborderWidth	**XmCBorderWidth**	**Dimension**	0	CSG
XmNcolormap	**XmCColormap**	**Colormap**	dynamic	CG
XmNdepth	**XmCDepth**	**int**	dynamic	CG
XmNdestroyCallback	**XmCCallback**	**XtCallbackList**	NULL	C
XmNheight	**XmCHeight**	**Dimension**	dynamic	CSG
XmNinitial-ResourcesPersistent	**XmCInitial-ResourcesPersistent**	**Boolean**	True	C
XmNmappedWhen-Managed	**XmCMappedWhen-Managed**	**Boolean**	True	CSG
XmNscreen	**XmCScreen**	**Screen** *	dynamic	CG
XmNsensitive	**XmCSensitive**	**Boolean**	True	CSG
XmNtranslations	**XmCTranslations**	**XtTranslations**	dynamic	CSG
XmNwidth	**XmCWidth**	**Dimension**	dynamic	CSG
XmNx	**XmCPosition**	**Position**	0	CSG
XmNy	**XmCPosition**	**Position**	0	CSG

Callback Information

The DtEditor widget has three callback functions of interest:

DtNtextSelectCallback and **DtNtextDeselectCallback**

The **DtNtextSelectCallback** and **DtNtextDeselectCallback** resources allow an application to enable and disable certain commands or menu items based on whether there is a selection. **DtNtextSelectCallback** specifies a function called whenever the selection becomes non-NULL (that is, some text is selected within the edit window), while **DtNtextDeselectCallback** specifies a function called whenever the selection becomes NULL (that is, no text is selected within the edit window). The reasons sent by the callbacks are DtEDITOR_TEXT_SELECT and DtEDITOR_TEXT_DESELECT.

XmNhelpCallback

An application that wishes to present help information to the user on the DtEditor widget and its dialogs should set the **XmNhelpCallback** resource and use the Reason field passed as part of **DtEditorHelpCallbackStruct** to set the contents of its Help dialog. A pointer to the following structure is passed to the **XmNHelpCallback** callback.

```
typedef struct {
        int reason,
        XEvent *event
} XmAnyCallbackStruct;
```

reason

Indicates why the callback was invoked. The possible reasons are:

DtEDITOR_HELP_EDIT_WINDOW
: The help request originated in the edit window.

DtEDITOR_HELP_FORMAT_DIALOG
: The help request originated in the Help button in the Format dialog.

DtEDITOR_HELP_FORMAT_LEFT_MARGIN
: The help request originated in the Left Margin field in the Format dialog.

DtEDITOR_HELP_FORMAT_RIGHT_MARGIN
: The help request originated in the Right Margin field in the Format dialog.

DtEDITOR_HELP_FORMAT_ALIGNMENT
: The help request originated in the Alignment buttons in the Format dialog.

DtEDITOR_HELP_CHANGE_DIALOG
: The help request originated in the Help button in the Find/Change dialog.

DtEDITOR_HELP_CHANGE_FIND
: The help request originated in the Find field in the Find/Change dialog.

DtEDITOR_HELP_CHANGE_CHANGE
: The help request originated in the Change To field in the Find/Change dialog.

event
: A pointer to the **XEvent** that caused this callback to be invoked. It may be NULL.

Translations

The DtEditor widget translations for the edit window are described in the following list. The **DtNtextTranslations** resource can be used to modify these translations.

KLeft
: *backward-character*( )

MShift KLeft
: *key-select*(*left*)

MCtrl KLeft
: *backward-word*( )

MShift MCtrl KLeft
: *backward-word*(*extend*)

KRight
: *forward-character*( )

MShift KRight
: *key-select*(*right*)

MCtrl KRight
: *forward-word*( )

MShift MCtrl KRight
: *forward-word*(*extend*)

KUp
: *process-up*( )

MShift KUp
process-shift-up( )

MCtrl KUp
backward-paragraph( )

MShift MCtrl KUp
backward-paragraph(*extend*)

KDown
process-down( )

MShift KDown
process-shift-down( )

MCtrl KDown
forward-paragraph( )

MShift MCtrl KDown
forward-paragraph(*extend*)

KBeginLine
beginning-of-line( )

MShift KBeginLine
beginning-of-line(*extend*)

KEndLine
end-of-line( )

MShift KEndLine
end-of-line(*extend*)

KPageUp
previous-page( )

MShift KPageUp
previous-page(*extend*)

KPageLeft
page-left( )

KPageDown
next-page( )

MShift KPageDown
next-page(*extend*)

KPageRight
page-right( )

KBeginData
beginning-of-file( )

MShift KBeginData
beginning-of-file(*extend*)

KEndData
end-of-file( )

MShift KEndData
end-of-file(*extend*)

KDelete
delete-next-character( )

MCtrl KDelete
delete-to-end-of-line( )

KBackSpace
delete-previous-character( )

MCtrl KBackSpace
delete-previous-word( )

MShift KBackSpace
delete-to-start-of-line( )

MAlt KBackSpace
undo-edit( )

MCtrl Kz
undo-edit( )

MCtrl K/
select-all( )

MCtrl K\
deselect-all( )

MCtrl Kq
quote-next-character( )

MCtrl Kx
cut-clipboard( )

MCtrl Kc
copy-clipboard( )

MCtrl Kv
paste-clipboard( )

KHelp
Help( )

KInsert
toggle-insert-mode( )

KEnter
new-line-and-indent( )

MAnyKCancel
process-cancel( )

Action Routines

The DtEditor widget action routines are described here:

backward-character( )
This action moves the insertion cursor one character to the left. This action may have different behaviour in a right-to-left language environment.

backward-paragraph(*extend*)
If this action is called with no argument, it moves the insertion cursor to the first non-whitespace character following the first previous blank line or beginning of the

text. If the insertion cursor is already at the beginning of a paragraph, the action moves the insertion cursor to the beginning of the previous paragraph.

If this action is called with an argument of *extend*, it moves the insertion cursor, as in the case of no argument, and extends the current selection.

backward-word(*extend*)
If this action is called with no argument, it moves the insertion cursor to the first non-whitespace character after the first whitespace character to the left or after the beginning of the line. If the insertion cursor is already at the beginning of a word, this action moves the insertion cursor to the beginning of the previous word. This action may have different behaviour in a locale other than the C locale.

If this action is called with an argument of *extend*, it moves the insertion cursor, as in the case of no argument, and extends the current selection.

beginning-of-file(*extend*)
If this action is called with no argument, it moves the insertion cursor to the beginning of the text.

If this action is called with an argument of *extend*, it moves the insertion cursor, as in the case of no argument, and extends the current selection.

beginning-of-line(*extend*)
If this action is called with no argument, it moves the insertion cursor to the beginning of the line.

If this action is called with an argument of *extend*, it moves the insertion cursor, as in the case of no argument, and extends the current selection.

clear-selection( )
This action clears the current selection by replacing each character except <carriage-return> with a <space> character.

copy-clipboard( )
This action copies the current selection to the clipboard.

cut-clipboard( )
This action cuts the current selection to the clipboard.

delete-next-character( )
If there is a non-NULL selection, this action deletes the selection; otherwise, it deletes the character following the insertion cursor.

delete-next-word( )
If there is a non-NULL selection, this action deletes the selection; otherwise, it deletes the characters following the insertion cursor to the next space, tab or end of line character.

delete-previous-character( )
If there is a non-NULL selection, this action deletes the selection; otherwise, it deletes the character of text immediately preceding the insertion cursor.

delete-previous-word( )
If there is a non-NULL selection, this action deletes the selection; otherwise, it deletes the characters preceding the insertion cursor to the next space, tab or beginning of the line character. This action may have different behaviour in a locale other than the C locale.

delete-to-end-of-line( )
If there is a non-NULL selection, this action deletes the selection; otherwise, it deletes the characters following the insertion cursor to the next end-of-line character.

delete-to-start-of-line( )
If there is a non-NULL selection, this action deletes the selection; otherwise, it deletes the characters preceding the insertion cursor to the previous beginning-of-line character.

deselect-all( )
This action deselects the current selection.

end-of-file(*extend*)
If this action is called with no argument, it moves the insertion cursor to the end of the text.

If this action is called with an argument of *extend*, it moves the insertion cursor, as in the case of no argument, and extends the current selection.

end-of-line(*extend*)
If this action is called with no argument, it moves the insertion cursor to the end of the line.

If this action is called with an argument of *extend*, it moves the insertion cursor, as in the case of no argument, and extends the current selection.

forward-character( )
This action moves the insertion cursor one character to the right. This action may have different behaviour in a right-to-left language environment.

forward-paragraph(*extend*)
If this action is called with no argument, it moves the insertion cursor to the first non-whitespace character following the next blank line. If the insertion cursor is already at the beginning of a paragraph, this action moves the insertion cursor to the beginning of the next paragraph.

If this action is called with an argument of *extend*, it moves the insertion cursor, as in the case of no argument, and extends the current selection.

forward-word(*extend*)
If this action is called with no argument, it moves the insertion cursor to the first whitespace character or end-of-line following the next non-whitespace character. If the insertion cursor is already at the end of a word, this action moves the insertion cursor to the end of the next word. This action may have different behaviour in a locale other than the C locale.

If called with an argument of *extend*, this action moves the insertion cursor, as in the case of no argument, and extends the current selection.

Help( )
This action calls the callbacks for **XmNhelpCallback** if any exist. If there are no help callbacks for this widget, this action calls the help callbacks for the nearest ancestor that has them.

insert-string(*string*)
This action deletes the entire selection if there is a non-NULL selection and the cursor is not disjoint from it. It inserts **string** before the insertion cursor.

key-select(*direction*)
If this action is called with an argument of *right*, it moves the insertion cursor one character to the right and extends the current selection. If this action is called with an argument of *left*, it moves the insertion cursor one character to the left and extends the current selection. If this action is called with no argument, it extends the current selection.

newline-and-backup( )
If there is a non-NULL selection and the cursor is not disjoint from it, this action deletes the entire selection, inserts a newline just before the insertion cursor and repositions the insertion cursor to the end of the line before the newline.

newline-and-indent( )
If there is a non-NULL selection and the cursor is not disjoint from it, this action deletes the entire selection, inserts a newline and then the same number of whitespace characters as at the beginning of the previous line.

next-page(*extend*)
If this action is called with no argument, it moves the insertion cursor forward one page.

If this action is called with an argument of *extend*, it moves the insertion cursor, as in the case of no argument, and extends the current selection.

page-left( )
This action scrolls the viewing window left one page of text.

page-right( )
This action scrolls the viewing window right one page of text.

paste-clipboard( )
This action pastes the contents of the clipboard before the insertion cursor.

previous-page(*extend*)
If this action is called with no argument, it moves the insertion cursor back one page.

If this action is called with an argument of *extend*, it moves the insertion cursor, as in the case of no argument, and extends the current selection.

process-cancel( )
This action cancels the current *extend-adjust*( ) or *secondary-adjust*( ) operation and leaves the selection state as it was before the operation; otherwise, (and if the parent is a manager) it passes the event to the parent.

process-down( )
This action moves the insertion cursor down one line.

process-shift-down( )
This action moves the insertion cursor down one line, extending the current selection.

process-shift-up( )
This action moves the insertion cursor up one line, extending the current selection.

process-up( )
This action moves the insertion cursor up one line.

select-all( )
Selects all text.

toggle-insert-mode( )

This action toggles the state of the text insertion mode. By default, characters typed into the DtEditor widget are inserted at the position of the insertion cursor. In overstrike mode, characters entered into the DtEditor widget replace the characters that directly follow the insertion cursor. In overstrike mode, characters are appended to the end of the line when the end of a line is reached.

quote-next-character( )

This action treats the next typed character as a special character and inserts it into the text without interpreting it. Enables the insertion of special instructional characters or special language characters, such as the character marking a form feed or the umlaut used in German text.

undo-edit( )

This action undoes the last change (deletion or insertion) made to the text. A change consists of either a set of consecutive insertions, or a set of consecutive deletions followed by up to one set of consecutive insertions. An insertion is consecutive if there have been no intervening deletions, and it is continuing forward from the same point. A deletion is consecutive if there have been no intervening insertions, and its start or end position is coincidental with the last deletion (that is, the deletion is continuing from the same point, either forward or backward). Undoing an edit once restores the original text. Undoing an edit twice restores the last change.

SEE ALSO

<Dt/Editor.h>, *DtCreateEditor*( ), *DtEditorAppend*( ), *DtEditorAppendFromFile*( ), *DtEditorChange*( ), *DtEditorCheckForUnsavedChanges*( ), *DtEditorClearSelection*( ), *DtEditorCopyToClipboard*( ), *DtEditorCutToClipboard*( ), *DtEditorDeleteSelection*( ), *DtEditorDeselect*( ), *DtEditorFind*( ), *DtEditorFormat*( ), *DtEditorGetContents*( ), *DtEditorGetInsertionPosition*( ), *DtEditorGetLastPosition*( ), *DtEditorGetSizeHints*( ), *DtEditorGoToLine*( ), *DtEditorInsert*( ), *DtEditorInsertFromFile*( ), *DtEditorInvokeFindChangeDialog*( ), *DtEditorInvokeFormatDialog*( ), *DtEditorPasteFromClipboard*( ), *DtEditorReplace*( ), *DtEditorReplaceFromFile*( ), *DtEditorSaveContentsToFile*( ), *DtEditorSelectAll*( ), *DtEditorSetContents*( ), *DtEditorSetContentsFromFile*( ), *DtEditorSetInsertionPosition*( ), *DtEditorTraverseToEditor*( ), *DtEditorUndoEdit*( ); *Composite*, *Constraint*, *Core*, *XmBulletinBoard*, *XmFontList*, *XmForm*, *XmManager* in the X/Open CAE Specification, **Motif Toolkit API**.

CHANGE HISTORY

First released in Issue 1.

9.3 Functions

This section defines the functions, macros and external variables that provide XCDE text editing services to support application portability at the C-language source level.

NAME

DtCreateEditor — create a new instance of a DtEditor widget

SYNOPSIS

```
#include <Dt/Editor.h>

Widget DtCreateEditor(Widget parent,
                      String name,
                      ArgList arglist,
                      Cardinal argcount);
```

DESCRIPTION

The *DtEditorCreate*( ) function creates an instance of a DtEditor widget and returns the associated widget ID.

The *parent* argument specifies the parent widget ID.

The *name* argument specifies the name of the created widget.

The *arglist* argument specifies the argument list.

The *argcount* argument specifies the number of attribute and value pairs in the argument list (*arglist*).

RETURN VALUE

Upon successful completion, the *DtEditorCreate*( ) function returns the DtEditor widget ID; otherwise, it returns NULL.

SEE ALSO

<Dt/Editor.h>, *DtEditor*( ).

CHANGE HISTORY

First released in Issue 1.

NAME

DtEditorAppend — append data to a DtEditor widget

SYNOPSIS

```
#include <Dt/Editor.h>

DtEditorErrorCode DtEditorAppend(Widget widget,
                                 DtEditorContentRec *data);
```

DESCRIPTION

The *DtEditorAppend*( ) function appends either a NULL-terminated string, wide character string or sized buffer after the last character in a DtEditor widget. The data is transferred to the DtEditor widget using a **DtEditorContentRec**, which indicates the type of data being transferred along with the actual data. After the data is appended, the insertion cursor is positioned at the new last character.

The *widget* argument specifies the DtEditor widget ID.

The *data* argument points to the data structure containing the data to append.

For a complete definition of **DtEditorContentRec**, see **<Dt/Editor>**.

RETURN

Upon successful completion, the *DtEditorAppend*( ) function returns DtEDITOR_NO_ERRORS; otherwise, it returns one of the following values:

DtEDITOR_INVALID_TYPE
: The *type* field is unrecognised.

DtEDITOR_ILLEGAL_SIZE
: The size of the buffer passed in is negative.

DtEDITOR_NULL_ITEM
: The buffer is NULL.

EXAMPLES

The following code segment sets the contents of a DtEditor widget to ‘‘The quick brown fox.’’

```
Widget                  editor;
DtEditorContentRec      cr;
DtEditorErrorCode       status;
char                    *sampleString1="The quick",
                        *secondString2=" brown fox";

cr.type = DtEDITOR_TEXT;
cr.value.string = sampleString1;
status = DtEditorSetContents(editor, &cr);
if (status != DtEDITOR_NO_ERRORS) {
        printf("Unable to set the contents of the widget\n");
} else {
        cr.type = DtEDITOR_TEXT;
        cr.value.string = sampleString2;
        status = DtEditorAppend(editor, &cr);
        if (status != DtEDITOR_NO_ERRORS)
                printf("Unable to append to the contents of the widget\n");
}
```

APPLICATION USAGE

If the data is in a disk file, rather than in memory, the application should use *DtEditorAppendFromFile*( ).

SEE ALSO

<Dt/Editor.h>, *DtEditor*( ), *DtEditorAppendFromFile*( ), *DtEditorGetContents*( ), *DtEditorInsert*( ), *DtEditorInsertFromFile*( ), *DtEditorReplace*( ), *DtEditorReplaceFromFile*( ), *DtEditorSaveContentsToFile*( ), *DtEditorSetContents*( ), *DtEditorSetContentsFromFile*( ).

CHANGE HISTORY

First released in Issue 1.

NAME

DtEditorAppendFromFile — append data from a file into a DtEditor widget

SYNOPSIS

```
#include <Dt/Editor.h>

DtEditorErrorCode DtEditorAppendFromFile(Widget widget,
                                         char *fileName);
```

DESCRIPTION

DtEditorAppendFromFile( ) function appends data from a file to after the last character in a DtEditor widget. After the data is appended, the insertion cursor is positioned at the new last character.

The *widget* argument specifies the DtEditor widget ID.

The *fileName* argument is the pathname of the file relative to the local system.

RETURN VALUE

Upon successful completion, the *DtEditorAppendFromFile*( ) function returns one of the following values:

DtEDITOR_NO_ERRORS
: The file is readable and writable.

DtEDITOR_READ_ONLY_FILE
: The file is read only.

Otherwise, if the *DtEditorAppendFromFile*( ) function cannot append the data into the DtEditor widget, it returns one of the following values:

DtEDITOR_NONEXISTENT_FILE
: The file does not exist.

DtEDITOR_DIRECTORY
: The file is a directory.

DtEDITOR_CHAR_SPECIAL_FILE
: The file is a character-special device.

DtEDITOR_BLOCK_MODE_FILE
: The file is a block-mode device.

DtEDITOR_NO_FILE_ACCESS
: The file cannot be accessed.

DtEDITOR_UNREADABLE_FILE
: The file is unreadable for an unspecified reason.

APPLICATION USAGE

If the data is in memory, rather than a disk file, the application should use *DtEditorAppend*( ).

SEE ALSO

<Dt/Editor.h>, *DtEditor*( ), *DtEditorAppend*( ). *DtEditorGetContents*( ), *DtEditorInsert*( ), *DtEditorInsertFromFile*( ), *DtEditorReplace*( ), *DtEditorReplaceFromFile*( ), *DtEditorSaveContentsToFile*( ), *DtEditorSetContents*( ), *DtEditorSetContentsFromFile*( ).

CHANGE HISTORY

First released in Issue 1.

NAME

DtEditorChange — change one or all occurrences of a string in a DtEditor widget

SYNOPSIS

```
#include <Dt/Editor.h>

Boolean DtEditorChange(Widget widget,
                       DtEditorChangeValues *findChangeStrings,
                       unsigned int instanceToChange);
```

DESCRIPTION

The *DtEditorChange*( ) function replaces the next occurrence of a string, all occurrences of the string, or the currently selected text in a DtEditor widget with a replacement string. The string to search for and the value to change it to can be the last values entered in the Find/Change dialog (see *DtEditorInvokeFindChangeDialog*( )) or passed as arguments to *DtEditorChange*( ).

The search begins at the insertion cursor. If the string is not found by the time the end of the document is reached, the search continues at the beginning of the document, stopping at the character before the insertion cursor.

The *widget* argument specifies the DtEditor widget ID.

The *findChangeStrings* argument specifies the string to change and the replacement value. If *findChangeStrings* is NULL, *DtEditorChange*( ) uses the last string specified in the Find and Change To fields of the Find/Change dialog. If the *instanceToChange* argument is DtEDITOR_CURRENT_SELECTION, the Find field of **DtEditorChangeValues** is ignored.

If the *instanceToChange* argument is set to DtEDITOR_NEXT_OCCURRENCE, *DtEditorChange*( ) replaces the next occurrence (relative to the insertion cursor) of the find string. If this argument is set to DtEDITOR_ALL_OCCURRENCES, all instances of the find string are changed. If this argument ia set to DtEDITOR_CURRENT_SELECTION, the Find field of **DtEditorChangeValues**, is ignored and the currently selected text is replaced.

For a complete definition of **DtEditorChangeValues**, see **<Dt/Editor.h>**.

RETURN VALUE

Upon successful completion, the *DtEditorChange*( ) function returns True if the substitution occurred; otherwise, it returns False.

SEE ALSO

<Dt/Editor.h>, *DtEditor*( ), *DtEditorFind*( ), *DtEditorInvokeFindChangeDialog*( ).

CHANGE HISTORY

First released in Issue 1.

NAME

DtEditorCheckForUnsavedChanges — report whether text has been edited

SYNOPSIS

```
#include <Dt/Editor.h>

Boolean DtEditorCheckForUnsavedChanges(Widget widget);
```

DESCRIPTION

The *DtEditorCheckForUnsavedChanges*( ) function reports whether the text contained in the edit window of a DtEditor widget has been modified since the last call to *DtEditorGetContents*( ) or *DtEditorSaveContentsToFile*( ), including inserting, deleting or moving text with the keyboard or mouse. For information about retrieving the text without affecting whether *DtEditorCheckForUnsavedChanges*( ) reports that all changes have been saved, see *DtEditorGetContents*( ) and *DtEditorSaveContentsToFile*( ).

The *widget* argument specifies the DtEditor widget ID.

RETURN VALUE

Upon successful completion, the *DtEditorCheckForUnsavedChanges*( ) function returns True if changes have been made to the contents of the DtEditor widget since the last call to *DtEditorGetContents*( ) or *DtEditorSaveContentsToFile*( ); otherwise, it returns False.

SEE ALSO

<Dt/Editor.h>, *DtEditor*( ), *DtEditorGetContents*( ), *DtEditorSaveContentsToFile*( ).

CHANGE HISTORY

First released in Issue 1.

NAME

DtEditorClearSelection — clear the primary selection in a DtEditor widget

SYNOPSIS

```
#include <Dt/Editor.h>

Boolean DtEditorClearSelection(Widget widget);
```

DESCRIPTION

The *DtEditorClearSelection*( ) function replaces the primary selection in a DtEditor widget, specified by *widget*, with blanks and newlines. Text can be selected and deselected programmatically with *DtEditorSelectAll*( ) and *DtEditorDeselect*( ).

The *widget* argument specifies the DtEditor widget ID.

RETURN VALUE

Upon successful completion, the *DtEditorClearSelection*( ) function returns True; otherwise, if the primary selection is NULL, or if the widget does not own the primary selection, it returns False.

SEE ALSO

<Dt/Editor.h>, *DtEditor*( ), *DtEditorCopyToClipboard*( ), *DtEditorCutToClipboard*( ), *DtEditorDeselect*( ), *DtEditorDeleteSelection*( ), *DtEditorPasteFromClipboard*( ), *DtEditorSelectAll*( ), *DtEditorUndoEdit*( ).

CHANGE HISTORY

First released in Issue 1.

NAME

DtEditorCopyToClipboard — copy the primary selection in a DtEditor widget to the clipboard

SYNOPSIS

```
#include <Dt/Editor.h>

Boolean DtEditorCopyToClipboard(Widget widget);
```

DESCRIPTION

The *DtEditorCopyToClipboard*( ) function copies to the clipboard the currently selected text in the DtEditor widget specified by the *widget* argument. Text can be selected and deselect programmatically with *DtEditorSelectAll*( ) and *DtEditorDeselect*( ).

The *widget* argument specifies the DtEditor widget ID.

RETURN VALUE

Upon successful completion, the *DtEditorCopyToClipboard*( ) function returns True; otherwise, if the primary selection is NULL, or if the widget does not own the primary selection, or if the function is unable to gain ownership of the clipboard selection, it returns False.

SEE ALSO

<Dt/Editor.h>, *DtEditor*( ), *DtEditorClearSelection*( ), *DtEditorCutToClipboard*( ), *DtEditorDeleteSelection*( ), *DtEditorDeselect*( ), *DtEditorPasteFromClipboard*( ), *DtEditorSelectAll*( ), *DtEditorUndoEdit*( ).

CHANGE HISTORY

First released in Issue 1.

NAME

DtEditorCutToClipboard — copy the primary selection in a DtEditor widget to the clipboard and delete the selected text

SYNOPSIS

```
#include <Dt/Editor.h>

Boolean DtEditorCutToClipboard(Widget widget);
```

DESCRIPTION

The *DtEditorCutToClipboard*( ) function copies the primary selected text in the DtEditor widget, specified by the *widget*, argument to the clipboard and then deletes the primary selected text. Text can be selected and deselected programmatically with *DtEditorSelectAll*( ) and *DtEditorDeselect*( ).

The *widget* argument specifies the DtEditor widget ID.

RETURN VALUE

Upon successful completion, the *DtEditorCutToClipboard*( ) function returns True; otherwise, if the primary selection is NULL, or if the widget doesn't own the primary selection, or if the function is unable to gain ownership of the clipboard selection, it returns False.

SEE ALSO

<Dt/Editor.h>, *DtEditor*( ), *DtEditorClearSelection*( ), *DtEditorCopyToClipboard*( ), *DtEditorDeleteSelection*( ), *DtEditorDeselect*( ), *DtEditorPasteFromClipboard*( ), *DtEditorSelectAll*( ), *DtEditorUndoEdit*( ).

CHANGE HISTORY

First released in Issue 1.

NAME

DtEditorDeleteSelection — delete the primary selection in the DtEditor widget

SYNOPSIS

```
#include <Dt/Editor.h>

Boolean DtEditorDeleteSelection(Widget widget);
```

DESCRIPTION

The *DtEditorDeleteSelection*( ) function removes the currently highlighted data in a DtEditor widget. Any data following the deleted data is moved up. Text can be selected and deselect programmatically with *DtEditorSelectAll*( ) and *DtEditorDeselect*( ).

The *widget* argument Specifies the DtEditor widget ID.

RETURN VALUE

Upon successful completion, the *DtEditorDeleteSelection*( ) function returns True; otherwise, if the primary selection is NULL, or if the widget does not own the primary selection, it returns False.

SEE ALSO

<Dt/Editor.h>, *DtEditorClearSelection*( ), *DtEditorCopyToClipboard*( ), *DtEditorCutToClipboard*( ), *DtEditorDeselect*( ), *DtEditorPasteFromClipboard*( ), *DtEditorSelectAll*( ), *DtEditorUndoEdit*( ).

CHANGE HISTORY

First released in Issue 1.

NAME

DtEditorDeselect — deselect the current selection in a DtEditor widget

SYNOPSIS

```
#include <Dt/Editor.h>

Boolean DtEditorDeselect(Widget widget);
```

DESCRIPTION

The *DtEditorDeselect*( ) function deselects any currently selected text in a DtEditor widget. The entire contents of a DtEditor widget may be selected with *DtEditorSelectAll*( ).

The *widget* argument specifies the DtEditor widget ID.

RETURN VALUE

Upon successful completion, the *DtEditorDeselect*( ) function returns True; otherwise, if the primary selection is NULL, or if the widget does not own the primary selection, it returns False.

SEE ALSO

<Dt/Editor.h>, *DtEditor*( ), *DtEditorClearSelection*( ), *DtEditorCopyToClipboard*( ), *DtEditorCutToClipboard*( ), *DtEditorDeleteSelection*( ), *DtEditorSelectAll*( ).

CHANGE HISTORY

First released in Issue 1.

NAME

DtEditorFind — search for the next occurrence of a string in a DtEditor widget

SYNOPSIS

```
#include <Dt/Editor.h>

Boolean DtEditorFind(Widget widget,
                     char *find);
```

DESCRIPTION

The *DtEditorFind*( ) function searches for the next occurrence (relative to the insertion cursor) of a string. The string is either the last find string value specified in the Find / Change dialog (see *DtEditorInvokeFindChangeDialog*( )) or is passed in as an argument.

The *widget* argument specifies the DtEditor widget ID.

The *find* argument specifies the string to search for. If *find* is NULL, *DtEditorFind*( ) uses the last string specified in the Find field of the Find / Change dialog. If the string is not found by the time the end of the document is reached, the search continues at the beginning of the document, stopping at the character before the insertion cursor.

RETURN VALUE

Upon successful completion, the *DtEditorFind*( ) function returns True if the search string was found; otherwise, it returns False.

SEE ALSO

<Dt/Editor.h>, *DtEditor*( ), *DtEditorChange*( ), *DtEditorInvokeFindChangeDialog*( ).

CHANGE HISTORY

First released in Issue 1.

NAME

DtEditorFormat — format all or part of the contents of a DtEditor widget

SYNOPSIS

```
#include <Dt/Editor.h>

DtEditorErrorCode DtEditorFormat(Widget widget,
                                 DtEditorFormatSettings *formatSettings,
                                 unsigned int amountToFormat);
```

DESCRIPTION

The *DtEditorFormat*( ) function formats all or part of the contents of the DtEditor widget according to the current text format settings in the Format Settings dialog. These options specify which margins and alignments (left aligned, right aligned, justified or centered) are used. Optionally, alternative settings can be passed as an argument to *DtEditorFormat*( ) in a data structure. This function formats either the paragraph containing the insertion cursor or the entire contents of the DtEditor widget, depending on the value of the *amountToFormat* argument.

The Format Settings dialog is displayed with *DtEditorInvokeFormatDialog*( ). For a complete description of formatting and the Format Settings dialog, see *DtEditor*.

The *widget* argument specifies the editor widget ID.

The *formatSettings* argument specifies left margin value, right margin value and the justification style. The LeftMargin and RightMargin fields of **DtEditorFormatSettings** must be zero or larger. The Alignment field can have a value of DtEDITOR_ALIGN_CENTER, DtEDITOR_ALIGN_JUSTIFY, DtEDITOR_ALIGN_LEFT or DtEDITOR_ALIGN_RIGHT. If the *formatSettings* argument is NULL, *DtEditorFormat*( ) uses the last format settings specified in the Format Settings dialog.

When the *amountToFormat* argument is set to DtEDITOR_FORMAT_ALL, it reformats all the text in the edit window. When this argument is set to DtEDITOR_PARAGRAPH, only the paragraph containing the insertion cursor is formatted.

For a complete definition of **DtEditorFormatSettings**, see **<Dt/Editor.h>**.

RETURN VALUE

Upon successful completion, the *DtEditorFormat*( ) function returns DtEDITOR_NO_ERRORS; otherwise, it returns one of the following values:

DtEDITOR_ILLEGAL_SIZE
: The left or right margin values are negative.

DtEDITOR_INVALID_RANGE
: The *amountToFormat* argument is not recognised.

DtEDITOR_INVALID_TYPE
: The Alignment field is not recognised.

SEE ALSO

<Dt/Editor.h>, *DtEditor*( ), *DtEditorInvokeFormatDialog*( ); *tmpnam*( ) in the X / Open CAE Specification, **System Interfaces and Headers**, **Issue 4, Version 2**.

CHANGE HISTORY

First released in Issue 1.

NAME

DtEditorGetContents — retrieve the contents of a DtEditor widget

SYNOPSIS

```
#include <Dt/Editor.h>

DtEditorErrorCode DtEditorGetContents(Widget widget,
                                      DtEditorContentRec *data,
                                      Boolean hardCarriageReturns,
                                      Boolean markContentsAsSaved);
```

DESCRIPTION

The *DtEditorGetContents*( ) function retrieves the entire contents of a DtEditor widget as a string, wide character string or sized buffer of data. The data is transferred from the DtEditor widget using a **DtEditorContentRec**, which indicates the type of data being transferred along with the actual data. If desired, any soft line feeds (word wraps) can be replaced with <newline>s.

The DtEditor widget tracks whether its contents have changed since they were last saved or retrieved. Setting the *markContentsAsSaved* argument to True retrieves a copy of the data without affecting whether *DtEditorCheckForUnsavedChanges*( ) reports that there are unsaved changes. This is useful if the application needs a temporary copy of the contents.

The *widget* argument specifies the DtEditor widget ID.

The *data* argument is a pointer to a data structure to receive the contents of *widget*.

The *hardCarriageReturns* argument, if set to True, indicates that the DtEditor widget should replace any soft line feeds (word wraps) with <newline>s when saving the data. When *hardCarriageReturns* is set to False, any line wrapped because it reaches the right edge of the window is saved as one complete line.

The *markContentsAsSaved* argument, if set to True, causes the DtEditor widget to mark that all changes made to date have been saved. When *markContentsAsSaved* is set to False, the DtEditor widget does not change its status regarding unsaved changes.

For a complete definition of **DtEditorContentRec**, see **<Dt/Editor>**.

RETURN VALUE

Upon successful completion, the *DtEditorGetContents*( ) function returns DtEDITOR_-NO_ERRORS; otherwise, it returns DtEDITOR_INVALID_TYPE if the Type field is not recognised.

EXAMPLES

The following code segment retrieves the contents of a DtEditor widget, marking that all changes to date have been saved.

```
Widget                  editor;
DtEditorContentRec      cr;
DtEditorErrorCode       status;
Boolean                 markContentsAsSaved = True;

cr.type = DtEDITOR_TEXT;
status = DtEditorGetContents(editor, &cr, markContentsAsSaved);
if (status == DtEDITOR_NO_ERRORS)
        printf("The contents are:\n%s\n", cr.value.string);
else
        printf("Unable to retrieve contents of the widget\n");
```

APPLICATION USAGE

To write the data directly to a file, the application should use *DtEditorSaveContentsToFile*( ).

SEE ALSO

<Dt/Editor.h>, *DtEditor*( ), *DtEditorAppend*( ), *DtEditorAppendFromFile*( ), *DtEditorCheckForUnsavedChanges*( ), *DtEditorInsert*( ), *DtEditorInsertFromFile*( ), *DtEditorReplace*( ), *DtEditorReplaceFromFile*( ), *DtEditorSaveContentsToFile*( ), *DtEditorSetContents*( ), *DtEditorSetContentsFromFile*( ).

CHANGE HISTORY

First released in Issue 1.

NAME

DtEditorGetInsertionPosition — retrieve the position of the insert cursor in a DtEditor widget

SYNOPSIS

```
#include <Dt/Editor.h>

XmTextPosition DtEditorGetInsertionPosition(Widget widget);
```

DESCRIPTION

The *DtEditorGetInsertionPosition*( ) function accesses the current position of the insertion cursor in the DtEditor widget. The position is an integer number of characters from the beginning of the widget's text buffer. The first character position is zero. The position of the insertion cursor can be set with *DtEditorSetInsertionPosition*( ).

The *widget* argument specifies the DtEditor widget ID.

RETURN VALUE

Upon successful completion, the *DtEditorGetInsertionPosition*( ) function returns an **XmTextPosition** value that indicates the position of the insertion cursor in the text; otherwise, it returns NULL.

SEE ALSO

<Dt/Editor.h>, *DtEditor*( ), *DtEditorGetLastPosition*( ), *DtEditorSetInsertionPosition*( ).

CHANGE HISTORY

First released in Issue 1.

NAME

DtEditorGetLastPosition — retrieve the position of the last character in a DtEditor widget

SYNOPSIS

```
#include <Dt/Editor.h>

XmTextPosition DtEditorGetLastPosition(Widget widget);
```

DESCRIPTION

The *DtEditorGetLastPosition*( ) function accesses the last text position in the DtEditor widget. The position is an integer number of characters from the beginning of the widget's buffer. Any text added to the end of the buffer is added after this position. The first character position is zero. The last character position is equal to the number of characters contained in the widget.

The *widget* argument specifies the DtEditor widget ID.

RETURN VALUE

Upon successful completion, the *DtEditorGetLastPosition*( ) function returns an **XmTextPosition** value that indicates the last position in the text; otherwise, it returns NULL.

APPLICATION USAGE

The position information is given in terms of characters, which may differ from the byte position when multi-byte code sets are in use.

SEE ALSO

<Dt/Editor.h>, *DtEditor*( ), *DtEditorGetInsertionPosition*( ), *DtEditorSetInsertionPosition*( ).

CHANGE HISTORY

First released in Issue 1.

NAME

DtEditorGetSizeHints — retrieve sizing information from a DtEditor widget

SYNOPSIS

```
#include <Dt/Editor.h>

void DtEditorGetSizeHints(Widget widget,
                          XSizeHints *pHints);
```

DESCRIPTION

The *DtEditorGetSizeHints*( ) function retrieves the current sizing information from a DtEditor widget, allowing the application to compute appropriate size hints for the window manager.

The *widget* argument specifies the DtEditor widget ID.

The *pHints* argument is a pointer to an **XSizeHints** structure into which the current sizing information is placed. The fields in this structure do not have to contain any values when it is passed in.

Upon successful completion, the *DtEditorGetSizeHints*( ) function fills in the following fields of the **XSizeHints** structure: minimum width (*min_width*) and height (*min_height*); width (*width_inc*) and height (*height_inc*) increment; and base width (*base_width*) and height (*base_height*); otherwise, the structure is unchanged. The Flags field is set to:

```
PMinSize | PResizeInc | PBaseSize
```

RETURN VALUE

The *DtEditorGetSizeHints*( ) function returns no value.

EXAMPLES

The following code segment sets the resize increment and minimum window size properties for the application.

```
Widget          editor,
                application_shell;
Display         display;
XSizeHints      size_hints;
long            supplied_return;

XGetWMSizeHints(display, XtWindow(application_shell),
                 &size_hints, &supplied_return, XA_WM_NORMAL_HINTS);

DtEditorGetSizeHints(editor, &size_hints);

XSetWMSizeHints(display, XtWindow(application_shell),
                 &size_hints, XA_WM_NORMAL_HINTS);
```

SEE ALSO

<Dt/Editor.h>, *DtEditor*( ).

CHANGE HISTORY

First released in Issue 1.

NAME

DtEditorGoToLine — move the insert cursor for a DtEditor widget to a specified line

SYNOPSIS

```
#include <Dt/Editor.h>

void DtEditorGoToLine(Widget widget
                      int lineNumber);
```

DESCRIPTION

The *DtEditorGoToLine*( ) function moves the insert cursor for the DtEditor widget to the beginning of the line specified by the *lineNumber* argument. The cursor can be moved to the last line by specifying DtEDITOR_LAST_LINE as the line number. If the line is not currently on-screen, the contents for the DtEditor widget are scrolled to display the new insertion position.

The *lineNumber* argument is the number of the line in the file, counting from 1. If the *lineNumber* argument is less than 1, the insert cursor is placed at the beginning of the first line. If the argument is greater than the total number of lines, the cursor is placed at the last line of text.

The insert cursor can be moved to a specific character position with *DtEditorSetInsertionPosition*( ). The **DtNtopCharacter** resource can be used to control which line is displayed at the top of the DtEditor widget.

The *widget* argument specifies the DtEditor widget ID.

The *lineNumber* argument specifies the line number within the DtEditor widget.

RETURN VALUE

The *DtEditorGoToLine*( ) function returns no value.

SEE ALSO

<Dt/Editor.h>, *DtEditor*( ), *DtEditorSetInsertionPosition*( ).

CHANGE HISTORY

First released in Issue 1.

NAME

DtEditorInsert — insert data into a DtEditor widget

SYNOPSIS

```
#include <Dt/Editor.h>

DtEditorErrorCode DtEditorInsert(Widget widget,
                                 DtEditorContentRec *data);
```

DESCRIPTION

The *DtEditorInsert*( ) function inserts either a string, wide character string or sized buffer at the insertion cursor position in a DtEditor widget. The data is transferred to the DtEditor widget using a **DtEditorContentRec**, which indicates the type of data being transferred along with the actual data. After the data is appended, the insertion cursor is positioned after the last character inserted.

The *widget* argument specifies the DtEditor widget ID.

The *data* argument is a pointer to a data structure containing the data to insert.

For a complete definition of **DtEditorContentRec**, see **<Dt/Editor>**.

RETURN VALUE

Upon successful completion, the *DtEditorInsert*( ) function returns DtEDITOR_NO_ERRORS; otherwise, it returns one of the following values:

DtEDITOR_INVALID_TYPE
: The Type field is not recognised.

DtEDITOR_ILLEGAL_SIZE
: The size of the buffer passed in is negative.

DtEDITOR_NULL_ITEM
: The buffer is NULL.

EXAMPLES

The following code segment sets the contents of a DtEditor widget to ‘‘The quick brown fox.’’

```
Widget                  editor;
DtEditorContentRec      cr;
DtEditorErrorCode       status;
char                    *sampleString1="The brown fox",
                        *sampleString2=" quick";

cr.type = DtEDITOR_TEXT;
cr.value.string = sampleString1;
status = DtEditorSetContents(editor, &cr);
if (status != DtEDITOR_NO_ERRORS) {
        printf("Unable to set contents of the widget\n");
} else {
  /*
   * Move the insertion cursor so it is after the
   * letter 'e' in "The".
   */
        DtEditorSetInsertionCursorPosition(editor, 2);

        cr.type = DtEDITOR_TEXT;
        cr.data.string = sampleString2;
        status = DtEditorInsert(editor, &cr);
        if (status != DtEDITOR_NO_ERRORS)
          printf("Unable to insert into the contents of the widget\n");
}
```

APPLICATION USAGE

If the data is in a disk file, rather than in memory, the application should use *DtEditorInsertFromFile*( ).

SEE ALSO

<Dt/Editor.h>, *DtEditor*( ), *DtEditorAppend*( ), *DtEditorAppendFromFile*( ), *DtEditorGetContents*( ), *DtEditorInsertFromFile*( ), *DtEditorReplace*( ), *DtEditorReplaceFromFile*( ), *DtEditorSaveContentsToFile*( ), *DtEditorSetContents*( ), *DtEditorGetInsertionPosition*( ), *DtEditorSetInsertionPosition*( ), *DtEditorSetContentsFromFile*( ).

CHANGE HISTORY

First released in Issue 1.

NAME

DtEditorInsertFromFile — insert data from a file into a DtEditor widget

SYNOPSIS

```
#include <Dt/Editor.h>

DtEditorErrorCode DtEditorInsertFromFile(Widget widget,
                                         char *fileName)
```

DESCRIPTION

The *DtEditorInsertFromFile*( ) function inserts data from a file to the insertion cursor position in a DtEditor widget. After the data is inserted, the insertion cursor is positioned after the last character inserted.

The *widget* argument specifies the DtEditor widget ID.

The *fileName* argument is the pathname of the file relative to the local system.

RETURN VALUE

Upon successful completion, the *DtEditorInsertFromFile*( ) function returns one of the following values:

DtEDITOR_NO_ERRORS
: The file is readable and writable.

DtEDITOR_READ_ONLY_FILE
: The file is read only.

Otherwise, if it cannot insert the data into the DtEditor widget, *DtEditorInsertFromFile*( ) returns one of the following values:

DtEDITOR_NONEXISTENT_FILE
: The file does not exist.

DtEDITOR_DIRECTORY
: The file is a directory.

DtEDITOR_CHAR_SPECIAL_FILE
: The file is a character-special device.

DtEDITOR_BLOCK_MODE_FILE
: The file is a block-mode device.

DtEDITOR_NO_FILE_ACCESS
: The file cannot be accessed.

DtEDITOR_UNREADABLE_FILE
: The file is unreadable for an unspecified reason.

APPLICATION USAGE

If the data is in memory, rather than a disk file, the application should use *DtEditorInsert*( ).

SEE ALSO

<Dt/Editor.h>, *DtEditor*( ), *DtEditorAppend*( ), *DtEditorAppendFromFile*( ), *DtEditorGetContents*( ), *DtEditorInsert*( ), *DtEditorReplace*( ), *DtEditorReplaceFromFile*( ), *DtEditorSaveContentsToFile*( ), *DtEditorSetContents*( ), *DtEditorGetInsertionPosition*( ), *DtEditorSetInsertionPosition*( ), *DtEditorSetContentsFromFile*( ).

CHANGE HISTORY

First released in Issue 1.

NAME

DtEditorInvokeFindChangeDialog — display the DtEditor widget dialog for searching and replacing text

SYNOPSIS

```
#include <Dt/Editor.h>

void DtEditorInvokeFindChangeDialog(Widget widget);
```

DESCRIPTION

The *DtEditorInvokeFindChangeDialog*( ) function displays the Find / Change dialog for the DtEditor widget. This dialog enables a user to search for, and optionally replace, a string in the text for the DtEditor widget. It also allows the user to specify a replacement string, which can be substituted for either the next occurrence of the search string or all occurrences. The Find / Change dialog remains displayed until the user closes it. For a complete description of the Find / Change dialog see *DtEditor*.

Subsequent searches for the last search string entered can be made by calling *DtEditorFind*( ). Subsequent substitutions can be made with *DtEditorChange*( ).

The *widget* argument specifies the DtEditor widget ID.

RETURN VALUE

The *DtEditorInvokeFindChangeDialog* ( ) function returns no value.

SEE ALSO

<Dt/Editor.h>, *DtEditor*( ), *DtEditorChange*( ), *DtEditorFind*( ).

CHANGE HISTORY

First released in Issue 1.

NAME

DtEditorInvokeFormatDialog — display the DtEditor widget dialog for choosing formatting options

SYNOPSIS

```
#include <Dt/Editor.h>

void DtEditorInvokeFormatDialog(Widget widget);
```

DESCRIPTION

The *DtEditorInvokeFormatDialog*( ) function displays the Format Settings dialog of the DtEditor widget. This dialog enables a user to set the text formatting options: margins and text alignments (left aligned, right aligned, justified or centered). The dialog also provides the capability to format either the paragraph containing the insertion cursor or the entire contents of the DtEditor widget. The Format Settings dialog remains displayed until the user closes it. For a complete description of the Format Settings dialog, see *DtEditor*.

Text can be formatted programmatically with *DtEditorFormat*( ).

The *widget* argument specifies the DtEditor widget ID.

RETURN VALUE

The *DtEditorInvokeFormatDialog*( ) function returns no value.

SEE ALSO

<Dt/Editor.h>, *DtEditor*( ), *DtEditorFormat*( ).

CHANGE HISTORY

First released in Issue 1.

NAME

DtEditorPasteFromClipboard — insert the clipboard selection into a DtEditor widget

SYNOPSIS

```
#include <Dt/Editor.h>

Boolean DtEditorPasteFromClipboard(Widget widget);
```

DESCRIPTION

The *DtEditorPasteFromClipboard*( ) function inserts the clipboard selection before the insertion cursor of the DtEditor widget. If the insertion cursor is inside the current selection, the clipboard selection replaces the selected text. Text can be cut or copied to the clipboard with *DtEditorCutToClipboard*( ) and *DtEditorCopyToClipboard*( ). Text can be selected and deselected programmatically with *DtEditorSelectAll*( ) and *DtEditorDeselect*( ).

The *widget* argument specifies the DtEditor widget ID.

RETURN VALUE

Upon successful completion, the *DtEditorPasteFromClipboard*( ) function returns True; otherwise, if the widget does not own the primary selection, the function returns False.

SEE ALSO

<Dt/Editor.h>, *DtEditor*( ), *DtEditorClearSelection*( ), *DtEditorCutToClipboard*( ), *DtEditorCopyToClipboard*( ), *DtEditorDeleteSelection*( ), *DtEditorDeselect*( ), *DtEditorSelectAll*( ), *DtEditorUndoEdit*( ).

CHANGE HISTORY

First released in Issue 1.

NAME

DtEditorReplace — replace a portion of the contents of a DtEditor widget

SYNOPSIS

```
#include <Dt/Editor.h>

DtEditorErrorCode DtEditorReplace(Widget widget,
                                  XmTextPosition startPos,
                                  XmTextPosition endPos,
                                  DtEditorContentRec *data);
```

DESCRIPTION

The *DtEditorReplace*( ) function replaces part of the contents of a DtEditor widget with a string, a wide character string or sized buffer. The data is transferred to the DtEditor widget using a **DtEditorContentRec**, which indicates the type of data being transferred along with the actual data. All data following the start position and up to, but not including, the end position is replaced. If the start position and the end position are equal, the data is inserted after the end position. The character positions begin at zero and are numbered sequentially from the beginning of the text. After the replacement, the insertion cursor is positioned after the last character inserted.

The *widget* argument specifies the DtEditor widget ID.

The *startPos* argument specifies the starting character position of the portion to replace. The replacement begins at this character.

The *endPos* argument specifies the ending character position of the portion to replace. The replacement ends before this character.

The *data* argument is a pointer to the data structure containing the data to insert.

For a complete definition of **DtEditorContentRec**, see **<Dt/Editor.h>**.

RETURN VALUE

Upon successful completion, the *DtEditorReplace*( ) function returns DtEDITOR_NO_ERRORS; otherwise, if it cannot replace the string, the function returns one of the following values:

DtEDITOR_INVALID_TYPE
: The Type field is not recognised.

DtEDITOR_INVALID_RANGE
: The *startPos* argument is greater than the *endPos* argument.

DtEDITOR_ILLEGAL_SIZE
: The size of the buffer passed in is negative.

DtEDITOR_NULL_ITEM
: The data buffer is NULL.

EXAMPLES

The following code segment modifies the contents of a DtEditor widget to ‘‘The quick fox.’’

```
Widget                  editor;
DtEditorContentRec      cr;
DtEditorErrorCode       status;
XmTextPosition          start = (XmTextPosition) 4,
                        end = (XmTextPosition) 9;
char                    *sampleString1="The brown fox",
                        *sampleString2="quick";

cr.type = DtEDITOR_TEXT;
cr.value.string = sampleString1;
status = DtEditorSetContents(editor, &cr);
if (status != DtEDITOR_NO_ERRORS) {
        printf("Unable to set contents of the widget\n");
} else {
        cr.type = DtEDITOR_TEXT;
        cr.data.string = sampleString2;
        status = DtEditorReplace(editor, start, end, &cr);
        if (status != DtEDITOR_NO_ERRORS)
          printf("Unable to replace part of the widget contents\n");
}
```

APPLICATION USAGE

If the data is in a disk file, rather than in memory, the application should use *DtEditorReplaceFromFile*( ).

SEE ALSO

<Dt/Editor.h>, *DtEditor*( ), *DtEditorAppend*( ), *DtEditorAppendFromFile*( ), *DtEditorGetContents*( ), *DtEditorInsert*( ), *DtEditorInsertFromFile*( ), *DtEditorReplaceFromFile*( ), *DtEditorSaveContentsToFile*( ), *DtEditorSetContents*( ), *DtEditorSetContentsFromFile*( ).

CHANGE HISTORY

First released in Issue 1.

NAME

DtEditorReplaceFromFile — replace a portion of the contents of a DtEditor widget with the contents of a file

SYNOPSIS

```
#include <Dt/Editor.h>

DtEditorErrorCode DtEditorReplaceFromFile(Widget widget,
                                          XmTextPosition startPos,
                                          XmTextPosition endPos,
                                          char *fileName);
```

DESCRIPTION

The *DtEditorReplaceFromFile*( ) function replaces part of the contents of a DtEditor widget with the contents of a file. All data following the start position and up to, but not including, the end position is replaced. If the start position and the end position are equal, the data is inserted after the end position. The character positions begin at zero and are numbered sequentially from the beginning of the text. After the replacement, the insertion cursor is positioned after the last character inserted.

The *widget* argument specifies the DtEditor widget ID.

The *startPos* argument specifies the starting character position of the portion to replace. The replacement begins at this character.

The *endPos* argument specifies the ending character position of the portion to replace. The replacement ends before this character.

The *fileName* argument is the pathname of the file relative to the local system.

RETURN VALUE

Upon successful completion, the *DtEditorReplaceFromFile*( ) function returns one of the following values:

DtEDITOR_NO_ERRORS
: The file is readable and writable.

DtEDITOR_READ_ONLY_FILE
: The file is read only.

Otherwise, if it cannot insert the data into the DtEditor widget, the function returns one of the following values:

DtEDITOR_INVALID_RANGE
: The *startPos* argument is greater than the *endPos* argument.

DtEDITOR_NONEXISTENT_FILE
: The file does not exist.

DtEDITOR_DIRECTORY
: The file is a directory.

DtEDITOR_CHAR_SPECIAL_FILE
: The file is a character-special device.

DtEDITOR_BLOCK_MODE_FILE
: The file is a block-mode device.

DtEDITOR_NO_FILE_ACCESS
: The file cannot be accessed.

DtEDITOR_UNREADABLE_FILE
The file is unreadable for an unspecified reason.

APPLICATION USAGE

If the data is in memory, rather than a disk file, the application should use *DtEditorReplace*( ).

SEE ALSO

<Dt/Editor.h>, *DtEditor*( ), *DtEditorAppend*( ), *DtEditorAppendFromFile*( ), *DtEditorGetContents*( ), *DtEditorInsert*( ), *DtEditorInsertFromFile*( ), *DtEditorReplace*( ), *DtEditorSaveContentsToFile*( ), *DtEditorSetContents*( ), *DtEditorSetContentsFromFile*( ).

CHANGE HISTORY

First released in Issue 1.

NAME

DtEditorReset — reset a DtEditor widget to its default state

SYNOPSIS

```
#include <Dt/Editor.h>

void DtEditorReset(Widget widget);
```

DESCRIPTION

The *DtEditorReset*( ) function deletes the contents of a DtEditor widget, resets the undo edit function, clears the last string searched for plus the last replacement string.

The *widget* argument specifies the DtEditor widget ID.

RETURN VALUE

The *DtEditorReset*( ) function returns no value.

APPLICATION USAGE

The *DtEditorReset*( ) is analogous to destroying a DtEditor widget and creating a new one with the current resource settings. It is useful when reusing a DtEditor widget.

SEE ALSO

<Dt/Editor.h>, *DtEditor*( ).

CHANGE HISTORY

First released in Issue 1.

NAME

DtEditorSaveContentsToFile — save the contents of a DtEditor widget to a file

SYNOPSIS

```
#include <Dt/Editor.h>

DtEditorErrorCode DtEditorSaveContentsToFile(Widget  widget,
                                             char *fileName,
                                             Boolean overwriteIfExists,
                                             Boolean hardCarriageReturns,
                                             Boolean markContentsAsSaved);
```

DESCRIPTION

The *DtEditorSaveContentsToFile*( ) function saves the entire contents of the DtEditor widget to a file, optionally replacing soft line feeds (word wraps) with <newline>s. If the file does not exist and the directory has the correct write permissions, the file is created. If the file exists and the *overwriteIfExists* argument is set to True, the contents of the file are overwritten. If the file or its directory does not have the correct write permissions, an error is returned.

The DtEditor widget tracks whether its contents have changed since they were last saved or retrieved. If the *markContentsAsSaved* argument is set to False, a copy of the data is saved without affecting whether *DtEditorCheckForUnsavedChanges*( ) reports that there are unsaved changes. This is useful if the application needs to save a copy of the contents to a temporary file.

The *widget* argument specifies the DtEditor widget ID.

The *fileName* argument is the pathname of the file relative to the local system.

The *overwriteIfExists* argument, if set to True, causes *DtEditorSaveContentsToFile*( ) to save the widget contents even though the file specified by the *fileName* argument exists and has correct write permissions. If this argument is set to False, *DtEditorSaveContentsToFile*( ) returns DtEDITOR_WRITABLE_FILE.

The *hardCarriageReturns* argument, if set to True, indicates that the widget should replace any soft line feeds (word wraps) with <newline>s when saving the data. When this argument is set to False, any line wrapped because it reaches the right edge of the window, is saved as one complete line.

The *markContentsAsSaved* argument, when set to True, causes the DtEditor widget to mark that all changes made to date have been saved. When this argument is set to False, the DtEditor widget does not change its status regarding unsaved changes. If an error arises during the save, the status does not change, regardless of the value of the *markContentsAsSaved* argument.

RETURN VALUE

Upon successful completion, the *DtEditorSaveContentsToFile*( ) function returns DtEDITOR_-NO_ERRORS; otherwise, if it cannot save the data to the file, the function returns one of the following values:

DtEDITOR_INVALID_FILENAME
: No file was specified.

DtEDITOR_UNWRITABLE_FILE
: The application does not have write permission for the file or directory.

DtEDITOR_CHAR_SPECIAL_FILE
: The file is a device-special file.

DtEDITOR_BLOCK_MODE_FILE
: The file is a block-mode device.

DtEDITOR_NO_FILE_ACCESS
: The file cannot be accessed.

DtEDITOR_SAVE_FAILED
: The contents could not be saved for an unspecified reason.

DtEDITOR_WRITABLE_FILE
: The named files exist and the *overwriteIfExists* argument is set to False.

EXAMPLES

The following code segment saves the contents of a DtEditor widget to the local file, Foo, substituting <newline>s for soft line feeds. It also indicates that all changes to the contents of the widget have been saved.

```
Widget                  editor;
DtEditorErrorCode       status;
char                    *fname = "Foo";
Boolean                 overwrite = False,
                        hardReturns = True,
                        markContentsAsSaved = True;

status = DtEditorSaveContentsToFile(editor, fname, overwrite,
                                    hardReturns,
                                    markContentsAsSaved);

switch(status )
{
        case DtEDITOR_NO_ERRORS:
                break;

        case DtEDITOR_WRITABLE_FILE:
                printf("Save failed.  The file already exists.\n");
                break;

        default:
                printf("Could not save contents.\n");
                break;
}
```

APPLICATION USAGE

The application should use *DtEditorGetContents*( ) to retrieve the data in a memory buffer, rather than a disk file.

SEE ALSO

<Dt/Editor.h>, *DtEditor*( ), *DtEditorAppend*( ), *DtEditorAppendFromFile*( ), *DtEditorCheckForUnsavedChanges*( ), *DtEditorGetContents*( ), *DtEditorInsert*( ), *DtEditorInsertFromFile*( ), *DtEditorReplace*( ), *DtEditorReplaceFromFile*( ), *DtEditorSetContentsFromFile*( ), *DtEditorSetContents*( ).

CHANGE HISTORY

First released in Issue 1.

NAME

DtEditorSelectAll — select all text in a DtEditor widget

SYNOPSIS

```
#include <Dt/Editor.h>

Boolean DtEditorSelectAll(Widget widget);
```

DESCRIPTION

The *DtEditorSelectAll*( ) function selects all text in a DtEditor widget. Any current selection can be programmatically deselected with *DtEditorDeselect*( ).

The *widget* argument specifies the DtEditor widget ID.

RETURN VALUE

Upon successful completion, the *DtEditorSelectAll*( ) function returns True; otherwise, if it is unable to gain ownership of the clipboard selection it returns False.

SEE ALSO

<Dt/Editor.h>, *DtEditor*( ), *DtEditorClearSelection*( ), *DtEditorCopyToClipboard*( ), *DtEditorCutToClipboard*( ), *DtEditorDeleteSelection*( ), *DtEditorDeselect*( ).

CHANGE HISTORY

First released in Issue 1.

NAME

DtEditorSetContents — place data into a DtEditor widget

SYNOPSIS

```
#include <Dt/Editor.h>

DtEditorErrorCode DtEditorSetContents(Widget widget,
                                      DtEditorContentRec *data);
```

DESCRIPTION

The *DtEditorSetContents*( ) function places a NULL–terminated string, wide character string or sized buffer into a DtEditor widget. Any data currently in the DtEditor widget is lost. The data is transferred to the DtEditor widget using a **DtEditorContentRec**, which indicates the type of data being transferred along with the actual data. After the data is placed into the DtEditor widget, the insertion cursor is positioned at the first character.

The *widget* argument specifies the DtEditor widget ID.

The *data* argument is a pointer to a data structure containing the new contents of *widget*.

For a complete definition of **DtEditorContentRec**, see **<Dt/Editor>**.

RETURN VALUE

Upon successful completion, the *DtEditorSetContents*( ) function returns DtEDITOR_-NO_ERRORS; otherwise, it returns one of the following values:

DtEDITOR_INVALID_TYPE
: The Type field is unrecognised.

DtEDITOR_ILLEGAL_SIZE
: The size of the buffer passed in is negative.

DtEDITOR_NULL_ITEM
: The buffer is NULL.

EXAMPLES

The following code segment sets the contents of a DtEditor widget to ‘‘The quick brown fox.’’

```
Widget                  editor;
DtEditorContentRec      cr;
DtEditorErrorCode       status;
char                    *sampleString="The quick brown fox";

cr.type = DtEDITOR_TEXT;
cr.value.string = sampleString;
status = DtEditorSetContents(editor, &cr);
if (status != DtEDITOR_NO_ERRORS)
        printf("Unable to set contents of the widget\n");
```

APPLICATION USAGE

If the data is in a disk file, rather than in memory, the application should use *DtEditorSetContentsFromFile*( ).

SEE ALSO

<Dt/Editor.h>, *DtEditor*( ), *DtEditorAppend*( ), *DtEditorAppendFromFile*( ), *DtEditorGetContents*( ), *DtEditorInsert*( ), *DtEditorInsertFromFile*( ), *DtEditorReplace*( ), *DtEditorReplaceFromFile*( ), *DtEditorSaveContentsToFile*( ), *DtEditorSetContentsFromFile*( ).

CHANGE HISTORY

First released in Issue 1.

NAME

DtEditorSetContentsFromFile — load data from a file into a DtEditor widget

SYNOPSIS

```
#include <Dt/Editor.h>

DtEditorErrorCode DtEditorSetContentsFromFile(Widget widget,
                                              char *fileName);
```

DESCRIPTION

The *DtEditorSetContentsFromFile*( ) function loads the contents of a file into a DtEditor widget. Any data currently in the DtEditor widget is lost. After the data is loaded, the insertion cursor is positioned at the first character.

The *widget* argument specifies the DtEditor widget ID.

The *fileName* argument is the pathname of the file relative to the local system.

RETURN VALUE

Upon successful completion, the *DtEditorSetContentsFromFile*( ) function returns one of the following values when it successfully loads the data into the DtEditor widget:

DtEDITOR_NO_ERRORS
: The file is readable and writable.

DtEDITOR_READ_ONLY_FILE
: The file is read only.

Otherwise, if it cannot load the data into the DtEditor widget, the function returns one of the following values:

DtEDITOR_NONEXISTENT_FILE
: The file does not exist.

DtEDITOR_DIRECTORY
: The file is a directory.

DtEDITOR_CHAR_SPECIAL_FILE
: The file is a character-special device.

DtEDITOR_BLOCK_MODE_FILE
: The file is a block-mode device.

DtEDITOR_NO_FILE_ACCESS
: The file cannot be accessed.

DtEDITOR_UNREADABLE_FILE
: The file is unreadable for an unspecified reason.

APPLICATION USAGE

If the data is in memory, rather than a disk file, the application should use *DtEditorSetContents*( ).

SEE ALSO

<Dt/Editor.h>, *DtEditor*( ), *DtEditorAppend*( ), *DtEditorAppendFromFile*( ), *DtEditorGetContents*( ), *DtEditorInsert*( ), *DtEditorInsertFromFile*( ), *DtEditorReplace*( ), *DtEditorReplaceFromFile*( ), *DtEditorSaveContentsToFile*( ), *DtEditorSetContents*( ).

CHANGE HISTORY

First released in Issue 1.

NAME

DtEditorSetInsertionPosition — set the position of the insert cursor in a DtEditor widget

SYNOPSIS

```
#include <Dt/Editor.h>

void DtEditorSetInsertionPosition(Widget widget,
                                  XmTextPosition position);
```

DESCRIPTION

The *DtEditorSetInsertionPosition*( ) function sets the insertion cursor position of the DtEditor widget. The current position of the insertion cursor can be retrieved with *DtEditorGetInsertionPosition*( ). The last text position of the DtEditor widget can be retrieved with *DtEditorGetLastPosition*( ).

The *widget* argument specifies the DtEditor widget ID.

The *position* argument specifies the position of the insertion cursor. This is an integer number of characters from the beginning of the text buffer. The first character position is zero. Values greater than the last position place the cursor at the last position (that is, at the end of the text).

RETURN VALUE

The *DtEditorSetInsertionPosition*( ) function returns no value.

SEE ALSO

<Dt/Editor.h>, *DtEditor*( ), *DtEditorGetInsertionPosition*( ), *DtEditorGetLastPosition*( ).

CHANGE HISTORY

First released in Issue 1.

NAME

DtEditorTraverseToEditor — set keyboard traversal to the edit window of a DtEditor widget

SYNOPSIS

```
#include <Dt/Editor.h>

void DtEditorTraverseToEditor(Widget widget);
```

DESCRIPTION

The *DtEditorTraverseToEditor*( ) function causes the Motif keyboard traversal to be set to the edit window of a DtEditor widget.

The *widget* argument specifies the DtEditor widget ID.

RETURN VALUE

The *DtEditorTraverseToEditor*( ) function returns no value.

SEE ALSO

<Dt/Editor.h>, *DtEditor*( ).

CHANGE HISTORY

First released in Issue 1.

NAME

DtEditorUndoEdit — undo the last edit made to the text in a DtEditor widget

SYNOPSIS

```
#include <Dt/Editor.h>

Boolean DtEditorUndoEdit(Widget widget);
```

DESCRIPTION

The *DtEditorUndoEdit*( ) function undoes the last change (deletion or insertion) made to the text in a DtEditor widget. A change consists of either a set of consecutive insertions, or a set of consecutive deletions followed by up to one set of consecutive insertions. An insertion is consecutive if there have been no intervening deletions, and it is continuing forward from the same point. A deletion is consecutive if there have been no intervening insertions, and its start or end position is coincidental with the last deletion (that is, the deletion is continuing from the same point, either forward or backward). Undoing an edit once restores the original text. Undoing an edit twice restores the last change.

The *widget* argument specifies the DtEditor widget ID.

RETURN VALUE

Upon successful completion, the *DtEditorUndoEdit*( ) function returns True; otherwise, if there is no pending undo it returns False.

SEE ALSO

<Dt/Editor.h>, *DtEditor*( ).

CHANGE HISTORY

First released in Issue 1.

9.4 Headers

This section describes the contents of headers used by the XCDE text editing service functions, macros and external variables.

Headers contain the definition of symbolic constants, common structures, preprocessor macros and defined types. Each function in Section 9.3 specifies the headers that an application must include in order to use that function. In most cases only one header is required. These headers are present on an application development system; they do not have to be present on the target execution system.

NAME

Dt/Editor.h — editor widget definitions

SYNOPSIS

```
#include <Dt/Editor.h>
```

DESCRIPTION

The **<Dt/Editor.h>** header defines structures, enumerations and function prototypes for the Editor widget class.

The header defines the **DtEditorDataFormat** enumeration data type, with at least the following members:

DtEDITOR_TEXT
: The data being transferred is a pointer to a NULL–terminated string of characters (a **char** ∗).

DtEDITOR_WCHAR
: The data being transferred is a wide character string (a **wchar_t** ∗).

DtEDITOR_DATA
: The data being transferred is a sized buffer (a **DtEditor_DataObj**).

The header defines the following structure:

```
typedef struct {
        unsigned int length;
        void *buf;
} DtEditor_DataObj;
```

The *length* argument is the size in bytes of the data buffer. The *buf* argument is a pointer to the data buffer.

The **DtEditorContentRec** structure is used to transfer data between an application and Editor widget. It indicates the type of data being transferred along with the actual data.

```
typedef struct {
        DtEditorDataFormat type;
        union {
                char *string;
                wchar_t *wchar;
                DtEditor_DataObj data;
        } value;
} DtEditorContentRec;
```

The *type* argument indicates the type of data contained in the structure. The *string* argument points to a NULL–terminated string of characters. It is valid when *type* is DtEDITOR_TEXT. The *wchar* argument points to a wide character string. It is valid when *type* is DtEDITOR_WCHAR. The *data* argument is a **DtEditor_DataObj** that contains the size of the data and a pointer to it. It is valid when *type* is DtEDITOR_DATA.

The **DtEditorChangeValues** structure is used optionally to specify the string to search for and its replacement value for *DtEditorChange*( ).

```
typedef struct {
        char *find,
        char *changeTo,
} DtEditorChangeValues;
```

The *find* argument is a text string to locate in an DtEditor widget. The *changeTo* argument is the replacement string for the one or more occurrences of the string specified in *find*. It can be NULL.

The **DtEditorFormatSettings** structure is used to optionally specify the left margin setting, right margin setting, and alignment style for *DtEditorFormat*( ).

```
typedef struct {
        int leftMargin,
        int rightMargin,
        unsigned int alignment,
} DtEditorFormatSettings;
```

The *leftMargin* argument is the column number of the left boundary when formatting text. Text is not extended to the left of this column. It must be non-negative. The *rightMargin* argument is the column number of the right boundary when formatting text. Text is not extended to the right of this column. It must be larger than *leftMargin*. The *alignment* argument specifies the style of alignment when formatting text (see the constants listed in this header).

The header declares the following variable:

```
WidgetClass      dtEditorWidgetClass;
```

The header defines the following constants for use with the *DtEditorChange*( ) function:

DtEDITOR_ALL_OCCURRENCES
: Change all instances of the find string.

DtEDITOR_CURRENT_SELECTION
: Replace the currently selected text with the replacement string.

DtEDITOR_NEXT_OCCURRENCE
: Change the next occurrence of the find string after the insertion cursor.

The header defines the following constants for use with the *DtEditorFormat*( ) function:

DtEDITOR_FORMAT_ALL
: Reformats all the text in a DtEditor.

DtEDITOR_FORMAT_PARAGRAPH
: Reformats only the paragraph containing the insertion cursor.

DtEDITOR_ALIGN_CENTER
: Centers each line of text between the left and right margins.

DtEDITOR_ALIGN_JUSTIFY
: Aligns the text flush with both the left and right margins.

DtEDITOR_ALIGN_LEFT
: Aligns the text flush with the left margin.

DtEDITOR_ALIGN_RIGHT
: Aligns the text flush with the right margin.

The header defines the following constant for use with the *DtEditorGoToLine*( ) function:

DtEDITOR_LAST_LINE
: Moves the cursor to the beginning of the last line in the widget.

The header defines the following **DtEditorErrorCode** constants:

DtEDITOR_NO_ERRORS
: The function completed its task without errors.

DtEDITOR_INVALID_TYPE
: The specified type is not a recognised **DtEditorDataFormat** when setting or retrieving contents or the specified Alignment type is not recognised when formatting text.

DtEDITOR_INVALID_RANGE
: The starting character position in a text replacement is greater than the ending character position or the specified Amount To Format when formatting text is not recognised.

DtEDITOR_NULL_ITEM
: The data buffer is NULL when passing data in a buffer.

DtEDITOR_ILLEGAL_SIZE
: The specified size of a data buffer is negative when passing data in a buffer, or the left and right margin values are illegal when formatting text.

DtEDITOR_INVALID_FILENAME
: No file was specified.

DtEDITOR_NONEXISTENT_FILE
: The file specified for reading does not exist.

DtEDITOR_UNREADABLE_FILE
: The file specified is unreadable for an unspecified reason.

DtEDITOR_READ_ONLY_FILE
: The file is read only.

DtEDITOR_NO_FILE_ACCESS
: The file cannot be accessed.

DtEDITOR_DIRECTORY
: The file specified is a directory.

DtEDITOR_CHAR_SPECIAL_FILE
: The file specified is a character-special device.

DtEDITOR_BLOCK_MODE_FILE
: The file specified is a block-mode device.

DtEDITOR_UNWRITABLE_FILE
: The application does not have write permission for the file or directory.

DtEDITOR_WRITABLE_FILE
: The specified file exists and the *overwriteIfExists* flag is set to False.

DtEDITOR_SAVE_FAILED
: The contents of the widget could not be saved for an unspecified reason.

The header defines the following enumeration values as reasons for the **DtNtextSelectCallback**:

DtEDITOR_TEXT_SELECT
: Some text has been selected within the edit window (that is, the selection has become non-NULL).

DtEDITOR_TEXT_DESELECT
: No text is selected within the edit window (that is, the selection has become NULL).

The header defines the following constants as reasons for the **XmNhelpCallback**:

DtEDITOR_HELP_EDIT_WINDOW
: The help request originated in the edit window.

DtEDITOR_HELP_FORMAT_DIALOG
: The help request originated from the Help button in the Format Settings dialog.

DtEDITOR_HELP_FORMAT_LEFT_MARGIN
: The help request originated from the Left Margin field in the Format Settings dialog.

DtEDITOR_HELP_FORMAT_RIGHT_MARGIN
: The help request originated from the Right Margin Field in the Format Settings dialog.

DtEDITOR_HELP_FORMAT_ALIGNMENT
: The help request originated from the Alignment buttons in the Format Settings dialog.

DtEDITOR_HELP_CHANGE_DIALOG
: The help request originated from the Help button in the Find / Change dialog.

DtEDITOR_HELP_CHANGE_FIND
: The help request originated from the Find field in the Find / Change dialog.

DtEDITOR_HELP_CHANGE_CHANGE
: The help request originated from the Change To field in the Find / Change dialog.

The header defines the following as functions:

```
Widget DtCreateEditor(Widget parent,
                      char *name,
                      ArgList arglist,
                      Cardinal argcount);

DtEditorErrorCode DtEditorAppend(Widget widget,
                                 DtEditorContentRec *data);

DtEditorErrorCode DtEditorAppendFromFile(Widget widget,
                                         char *fileName);

Boolean DtEditorChange(Widget widget,
                       DtEditorChangeValues *findChangeStrings,
                       unsigned int instanceToChange);

Boolean DtEditorCheckForUnsavedChanges(Widget widget);

Boolean DtEditorClearSelection(Widget widget);

Boolean DtEditorCopyToClipboard(Widget widget);

Boolean DtEditorCutToClipboard(Widget widget);

Boolean DtEditorDeleteSelection(Widget widget);

Boolean DtEditorDeselect(Widget widget);

Boolean DtEditorFind(Widget widget,
                     char * find);

DtEditorErrorCode DtEditorFormat(Widget widget,
                                 DtEditorFormatSettings *formatSettings,
                                 unsigned int amountToFormat);
```

```
DtEditorErrorCode DtEditorGetContents(Widget widget,
                                      DtEditorContentRec *data,
                                      BooleanhardCarriageReturns,
                                      Boolean markContentsAsSaved);

XmTextPosition DtEditorGetInsertionPosition(Widget widget);

XmTextPosition DtEditorGetLastPosition(Widget widget);

void DtEditorGetSizeHints(Widget widget,
                          XSizeHints *pHints);

void DtEditorGoToLine(Widget widget,
                      int lineNumber);

DtEditorErrorCode DtEditorInsert(Widget widget,
                                 DtEditorContentRec *data);

DtEditorErrorCode DtEditorInsertFromFile(Widget widget,
                                         char *fileName);

void DtEditorInvokeFindChangeDialog(Widget widget);

void DtEditorInvokeFormatDialog(Widget widget);

Boolean DtEditorPasteFromClipboard(Widget widget);

DtEditorErrorCode DtEditorReplace(Widget widget,
                                  XmTextPosition startPos,
                                  XmTextPosition endPos,
                                  DtEditorContentRec *data);

DtEditorErrorCode DtEditorReplaceFromFile(Widget widget,
                                          XmTextPosition startPos,
                                          XmTextPosition endPos,
                                          char *fileName);

DtEditorErrorCode DtEditorSaveContentsToFile(Widget widget,
                                             char *fileName,
                                             BooleanoverwriteIfExists,
                                             BooleanhardCarriageReturns,
                                             Boolean markContentsAsSaved);

Boolean DtEditorSelectAll(Widget widget);

DtEditorErrorCode DtEditorSetContents(Widget widget,
                                      DtEditorContentRec *data);

DtEditorErrorCode DtEditorSetContentsFromFile(Widget widget,
                                              char *fileName);

void DtEditorSetInsertionPosition(Widget widget,
                                  XmTextPosition position);

void DtEditorTraverseToEditor(Widget widget);

Boolean DtEditorUndoEdit(Widget widget);
```

CHANGE HISTORY

First released in Issue 1.

9.5 Command-Line Interfaces

This section defines the utility that provides XCDE text editing services.

NAME

dtpad — edit text files

SYNOPSIS

`dtpad [`*`–options`*`] [`*`file`*`]`

DESCRIPTION

The **dtpad** utility is a basic editor that supports editing text files in a manner consistent with other common Graphical User Interface text manipulation and file access mechanisms. Cursor positioning and text selection as well as access to various edit operations can be done via the standard Motif text manipulation mechanisms using the mouse or user-definable key combinations. Text can be cut, copied or pasted, or dragged to and from the Text Editor and/or other compliant application windows via the standard Motif Clipboard and ICCCM Primary and Secondary selection mechanisms. Also, standard dialogs are presented for accessing files and printing text.

The Text Editor also provides the following features:

- Undo of the previous edit operation.
- Search and replace.
- Simple formatting.

OPTIONS

The *dtpad* utility does not support the X/Open Utility Syntax Guidelines because it uses the X Window System convention of full-word options. The following options are available:

–saveOnClose

Automatically and silently saves the current text when there are unsaved changes and the Text Editor is closed. The default action for this situation posts a dialog asking whether or not to save the current text. This option inhibits the posting of the Save dialog when the Text Editor is closed. The Save dialog is always posted when a new file is specified and there are unsaved changes.

–missingFileWarning

Posts a Warning dialog whenever a file name is specified and the file does not exist or cannot be accessed.

–noReadOnlyWarning

Disables the Warning dialog posted whenever a file is specified for which the user does not have write permission. The default posts a Warning dialog whenever this situation occurs.

–noNameChange

Indicates that the default file name associated with the current text is not to change when the text is saved under a name different than what it was read in under. The current text can still be saved under a different file name; however, the default file name does not change. By default, the default file name is automatically changed to correspond to the last name under which the current text was saved.

–viewOnly

Disallows editing of text in the edit window, essentially turning the Text Editor into a text viewer. The default allows text editing in the edit window even if the text was obtained from a file for which the user does not have write permission.

OPERANDS

The following operand is supported:

file The file to be edited or viewed. If no *file* is specified, the Text Editor opens a new (empty) edit window and the file name must be specified when the contents are saved.

RESOURCES

Basic Resources			
Name	**Class**	**Type**	**Default**
saveOnClose	**SaveOnClose**	**Boolean**	False
missingFileWarning	**MissingFileWarning**	**Boolean**	False
readOnlyWarning	**ReadOnlyWarning**	**Boolean**	True
nameChange	**NameChange**	**Boolean**	True
viewOnly	**ViewOnly**	**Boolean**	False

saveOnClose

Indicates whether the Text Editor is to save automatically the current text when there are unsaved changes and the Text Editor is closed. Setting this resource to True automatically saves unsaved changes when the Text Editor is closed. This is equivalent to specifying the –**saveOnClose** command-line option.

missingFileWarning

Indicates whether a warning dialog is to be posted when a file is specified that does not exist or cannot be accessed. Setting this resource to True displays the warning. This is equivalent to specifying the –**missingFileWarning** command-line option.

readOnlyWarning

Indicates whether a warning dialog is to be posted when a file for which the user does not have write permission is read. Setting this resource to False suppresses the warning. This is equivalent to specifying the –**noReadOnlyWarning** command-line option.

nameChange

Indicates whether the current file name is to be changed when the current text is saved under a new name. Setting this resource to False does not allow the name to be reset. This is equivalent to specifying the –**noNameChange** command-line option.

viewOnly

Indicates whether text only be viewed or whether it can be edited in the edit window. Setting this resource to True disables text editing. This is equivalent to specifying the –**viewOnly** command-line option.

STDIN

Not used.

INPUT FILES

None.

ENVIRONMENT VARIABLES

The following environment variables affect the execution of *dtpad*:

DISPLAY Specify the default X Windows display to connect to.

LANG Provide a default value for the internationalisation variables that are unset or null. If *LANG* is unset or null, the corresponding value from the implementation-specific default locale will be used. If any of the internationalisation variables contains an invalid setting, the utility behaves as if none of the variables had been defined.

LC_ALL If set to a non-empty string value, override the values of all the other internationalisation variables.

LC_MESSAGES Determine the locale that is used to affect the format and contents of diagnostic messages written to standard error and informative messages written to standard output.

NLSPATH Determine the location of message catalogues for the processing of *LC_MESSAGES*.

ASYNCHRONOUS EVENTS

Default.

STDOUT

Not used.

STDERR

Not used.

OUTPUT FILES

None.

EXTENDED DESCRIPTION

None.

EXIT STATUS

The following exit values are returned:

0 Successful completion.

>0 An error occurred.

CONSEQUENCES OF ERRORS

Default.

APPLICATION USAGE

None.

EXAMPLES

None.

SEE ALSO

None.

CHANGE HISTORY

First released in Issue 1.

9.6 Actions

This section defines the actions that provide XCDE text editing services to support application portability at the C-language source or shell script levels.

NAME

dttextaction — XCDE text editing actions

SYNOPSIS

```
Dtpad [file]
Open file
Print file
TextEditor
```

DESCRIPTION

The XCDE Text Editing Services support the following text editing actions:

Dtpad
: Open an empty view of the desktop text editor.

Dtpad *file*
: Open a desktop text editor view of the text file named by the pathname in the *file* argument.

Open *file*
: Open a view of the text file named by the pathname in the *file* argument.

Print *file*
: Print the text file named by the pathname in the *file* argument.

TextEditor
: Open a view of the user's preferred text editor.

These actions can be invoked from an application using the *DtActionInvoke*( ) function or invoked from a command line using the *dtaction* utility.

SEE ALSO

dtaction, *DtActionInvoke*( ) in the **XCDI** specification.

CHANGE HISTORY

First released in Issue 1.

9.7 Messages

The XCDE text editing services support the ToolTalk Desktop and Media Exchange messages listed below for the following media types:

C_STRING
: Text in an arbitrary codeset

_DT_DATA
: Data that does not match any other data type

In addition, the Text Editor supports the messages below for any media type that does not have a specific editor registered.

The following messages are supported from the **XCDI** specification, **Section 6.6.2**, **Media Exchange Message Set**:

Instantiate
: Opens a new edit window for composing arbitrary file(s).

Edit Opens a new edit window for editing an existing file or buffer or for composing a specific new file or buffer.

Display
: Opens a new edit window for displaying an existing file or buffer.

The following messages are supported from the **XCDI** specification, **Section 6.6.1**, **Desktop Message Set**:

Quit Terminates the text editing services or closes a specific Text Editor edit window as specified by the *operation2Quit* argument. The *operation2Quit* argument must be the message ID of the Media Exchange request that created the edit window.

The default actions for notifying the user, saving or returning text and closing edit windows are:

- If *operation2Quit* is specified, the specified edit window is closed; otherwise, all edit window(s) are closed and the text editing services are terminated
- If there are unsaved changes, the user is notified and allowed to save the text and/or abort the *Quit*; otherwise, the user is not notified and the text is not saved (or returned if a buffer is being edited)

Both the *silent* and *force* arguments are supported. However, the semantics of *silent* differ from those described in **XCDI** specification, **Section 6.6.1**, **Desktop Message Set** in that the text editing services provides user notification only when there are unsaved changes, rather than user notification when an edit window is terminated. The following table describes variances in the default action for various combination of *silent* and *force*.

silent	***force***	***action***
False	**False**	*default*
True	**False**	If there are unsaved changes, the user is not notified, the text is not saved and the edit window is not terminated.
False	**True**	If there are unsaved changes, the user is still notified and allowed to save the text, but cannot abort the *Quit*.
True	**True**	If there are unsaved changes, the user is not notified, the text is not saved and the edit window is closed.

Whenever the *Quit* request is not carried out (i.e., in the default case when the user explicitly aborts the *Quit* or when *silent* is True and *force* is not specified or is False), the *Quit* request is failed with

Save Saves a specific edit window opened via an *Edit* request. The ID argument must have the **messageID** vtype and have the value of the message ID of the *Edit* request that created the edit window.

Saved
Sent when a file has been saved, as the result of a *Save* request or a user action.

9.8 Capabilities

A conforming implementation of the XCDE text editing services supports at least the following capabilities:

1. Provides text editing services as described in the following subsections.
2. Conforms to the Required items in the Application Style Checklist in Chapter 20 on page 355.
3. Has been internationalised using the standard interfaces in the X/Open CAE Specification, **System Interfaces and Headers**, **Issue 4, Version 2**, the X/Open CAE Specification, **Window Management: Xlib C Language Binding**, and the X/Open CAE Specification, **Motif Toolkit API**, and supports any locale available in the underlying X Window System Xlib implementation.

9.8.1 File Management

The following capabilities are supported for managing files:

1. The user can create a new document.
2. The user can open the document contained in an existing text file.
3. The user can save the document to a new file.
4. The user can save the document to the original text file.
5. The user can include an existing text file in a document. The contents of an included files is inserted into the current document at the location of the insertion cursor.
6. The user can drop file icons, buffer icons and selected text on the edit area. Such a drop is equivalent to an include of an existing file, but no file selection dialog is posted.
7. The user can drag selected text out of the edit area.

9.8.2 Presentation

The following capabilities are supported for presenting the text:

1. The editing area provides scroll bars to see text not visible in the window.
2. The user can find and optionally replace text in the document.
3. The user can print the current document.
4. The user can select any the following types of formatting for a single paragraph and for the whole document:

a. Align along the left margin

b. Align along the right margin

c. Align along both margins

d. Center each line

5. The user can set both left and right margins.
6. Formatting supports paragraph indents. The difference between the positions of the left-most characters of the first and second lines of a paragraph is the indent and is preserved for all lines of the paragraph.

9.8.3 Text Editing

The following capabilities are supported for editing text:

1. The user can cut text to the clipboard.
2. The user can copy text to the clipboard.
3. The user can paste text from the clipboard into the document.
4. The user can delete text from the document.
5. The user can replace text in the document with spaces. This is also called ‘‘clearing.’’
6. The user can undo the most recent edit operation.
7. The user can select all text in the document.
8. The user can select either insert mode or replace mode for text entry.
9. The user can copy and move text through drag and drop operations.

Chapter 10

Icon Editing Services

10.1 Introduction

The XCDE icon editing services allows users to create and modify icons that are used in the X/Open Common Desktop Environment. This service provides a graphical interface with a drawing area and tools and colours to use for creating icons and modifying previously created icons.

10.2 Actions

This section defines the actions that provide XCDE icon editing services to support application portability at the C-language source or shell script levels.

NAME

dticonaction — XCDE icon editing actions

SYNOPSIS

```
Dticon [icon]
Open icon
```

DESCRIPTION

The XCDE Icon Editing Services support the following icon editing actions:

Dticon

Open an empty icon editor view.

Dticon *icon*

Open an icon editor view of the bitmap or pixmap named by the pathname in the *icon* argument.

Open *icon*

Open an icon editor view of the bitmap or pixmap named by the pathname in the *icon* argument.

These actions can be invoked from an application using the *DtActionInvoke*( ) function or invoked from a command line using the *dtaction* utility.

SEE ALSO

dtaction, *DtActionInvoke*( ) in the **XCDI** specification.

CHANGE HISTORY

First released in Issue 1.

10.3 Messages

The XCDE icon editing services implement the *Edit* request for media types XPM and XBM. See the **XCDI** specification, **Section 6.6.2**, **Media Exchange Message Set**. These services also respond to the *Quit* desktop message. See the **XCDI** specification, **Section 6.6.1**, **Desktop Message Set**.

10.4 Capabilities

A conforming implementation of the XCDE icon editing services supports at least the following capabilities:

1. Conforms to the Required items in the Application Style Checklist in Chapter 20 on page 355, with the following exception that may exist on some implementations:
    a. There are certain operations that need not comply with checklist item 7-10. The option button in file selection dialogs allowing users to specify the format when saving a file need not be available.
2. Has been internationalised using the standard interfaces in the X/Open CAE Specification, **System Interfaces and Headers**, **Issue 4, Version 2**, the X/Open CAE Specification, **Window Management: Xlib C Language Binding**, and the X/Open CAE Specification, **Motif Toolkit API**, and supports any locale available in the underlying X Window System Xlib implementation.
3. Creates new icon files or modifies existing icon files.
4. Reads and writes XPM and XBM icon file formats; see Section 19.2 on page 353.
5. Provides a drawing area where the icon is created or modified.
6. Allows the drawing area to be a drop destination for icon files.
7. Provides geometric drawing operations, including line, polyline, rectangle, polygon, circle and ellipse, for creating or modifying the icon.
8. Supports the following operations within the drawing area:
    a. Copy, cut and paste the selected area
    b. Scale the selected area
    c. Rotate (left/right) the selected area
    d. Flip (vertical/horizontal) the selected area
9. Allows selection of the colour to use for drawing operations, including XCDE dynamically defined colours; see Section 19.2 on page 353.
10. Supports resizing of the icon width and height.
11. Allows specification of the hot spot within the icon.
12. Supports magnification of the icon within the drawing area.
13. Allows as input to the drawing area any portion of the display screen by providing an active bounding rectangle to the user for selecting the screen area to be used.

Chapter 11

GUI Scripting Services

11.1 Introduction

The XCDE GUI scripting services provide a means to develop interactive applications using the familiar shell programming environment defined in the X/Open CAE Specification, **Commands and Utilities**, **Issue 4, Version 2**. The GUI scripting services provide extensions to the shell that allow access to the variety of XCDE services.

11.2 Command-line Interface

This section defines the utility that provides XCDE GUI scripting services.

NAME

dtksh — shell command language interpreter with access to many X, Xt, Xm and XCDE functions

SYNOPSIS

```
dtksh [–abCefimnuvx] [–o option] [+abCefmnuvx] [+o option]
[command_file [argument...]]

dtksh [–abCefimnuvx] [–o option] [+abCefmnuvx] [+o option]
command_string [command_name [argument...]]

dtksh –s [–abCefimnuvx] [–o option] [+abeCefmnuvx] [+o option]
[argument...]]
```

DESCRIPTION

The *dtksh* utility is a version of the *sh* utility (defined in the X/Open CAE Specification, **Commands and Utilities**, **Issue 4, Version 2**) extended to support:

- Access to many X, Xt and Motif facilities from within a shell script
- Fully localised shell scripts
- Access to the XCDE application help system
- Customisation of script-based GUI attributes (such as font and colours) using the XCDE customisation tool
- Response to session-management **Save state** directives
- Response to window-management **Close** directives
- Access to most of the XCDE Desktop Services Message Set
- Access to many of the XCDE Data Typing API functions
- Access to the XCDE Action API functions

OPTIONS

See *sh* in the X/Open CAE Specification, **Commands and Utilities**, **Issue 4, Version 2**.

OPERANDS

See *sh* in the X/Open CAE Specification, **Commands and Utilities**, **Issue 4, Version 2**.

RESOURCES

The *dtksh* interpreter has no relevant resources outside of those that affect the various widgets that can be instantiated from within a *dtksh* script. Refer to the manual page of the relevant widget for information on the resources that apply to that widget.

STDIN

See *sh* in the X/Open CAE Specification, **Commands and Utilities**, **Issue 4, Version 2**.

INPUT FILES

See *sh* in the X/Open CAE Specification, **Commands and Utilities**, **Issue 4, Version 2**.

ENVIRONMENT VARIABLES

The following information describes the environment variables that *dtksh* uses that are in addition to those documented in the manual page for the *sh* command language interpreter.

Immediate Return Value (–)

Many of the category 3 commands (as described in **Return Values From Built-in Commands** on page 235) return a single value using an environment variable specified as the first argument to the command (in the synopses for these special commands, the first argument has the name *variable*). If this return value is immediately used in an expression, the special environment variable ‘‘–’’ can be used in place of a variable name. When *dtksh* encounters ‘‘–’’ as the name of the environment variable in which the return value is to be returned, it returns the result as the value of the command. This allows the shell script to embed the command call in another command call. (This feature works only for commands that return a single value; the value is the first argument and the argument has the name *variable*). For example:

```
XtDisplay DISPLAY $FORM
XSync $DISPLAY true
```

can be replaced by the equivalent:

```
XSync $(XtDisplay "–" $FORM) true
```

The reference to *$DISPLAY* is replaced with the value returned by the call to *XtDisplay*. This capability is available for all category 3 commands except those that create a widget, those that return more than a single value and those whose first argument is not named *variable*. Commands that do not accept ‘‘–’’ as the environment variable name include: *XtInitialize*, *XtCreateApplicationShell*, *XtCreatePopupShell*, *XtCreateManagedWidget* and *XtCreateWidget*; all commands of the form:

```
XmCreate...()
```

and most commands of the form:

```
tt_...()
```

Variables Set By XtInitialise

The *XtInitialize* command sets the following variables:

DTKSH_APPNAME
DTKSH_ARGV
DTKSH_TOPLEVEL

Callback Context Variables

An application registers a callback with a widget to specify which condition it is interested in, and what action should occur when that condition occurs. The action can be any arbitrary *dtksh* command line. For example:

```
XtAddCallback $WIDGET activateCallback "ActivateProc"
XtAddCallback $WIDGET activateCallback "XtSetSensitive $BUTTON false"
```

A callback needs to be passed some context so it can determine what condition led to its call. For a C procedure, this information is typically passed in a *call_data* structure. For example, a Scale widget invoking a **valueChangedCallback** passes in *call_data* an instance of the following structure:

```
typedef struct {
        int    reason;
        XEvent *event;
        int    value;
} XmScaleCallbackStruct;
```

The C application's callback does something like:

```
if (scaleCallData->reason == XmCR_VALUE_CHANGED) {
        eventType = scaleCallData->event->type;
        display = scaleCallData->event->xany.display;
}
```

Similarly in *dtksh*, when a callback is invoked, the following special environment variables are set up before the callback command executes:

CB_WIDGET
: Set to the widget handle for the widget invoking the callback.

CB_CALL_DATA
: Set to the address of the *call_data* structure passed by the widget to the callback, but its usefulness lies in the nested sub-variables associated with it.

The *CB_CALL_DATA* environment variable represents a pointer to a structure; access to its fields uses a syntax similar to the C code. Nested environment variables are defined, named the same as the fields of the structure (but folded to all upper case), and use a dot to indicate containment of an element in a structure. Thus, the preceding C code, to access the *call_data* provided by the Scale widget, translates to:

```
if [${CB_CALL_DATA.REASON} = "CR_VALUE_CHANGED"]; then
        eventType=${CB_CALL_DATA.EVENT.TYPE}
        display=${CB_CALL_DATA.EVENT.XANY.DISPLAY}
fi
```

The same is true of the event structure within the *call_data* structure.

For most callback structures, the shell script is able to reference any of the fields defined for the particular callback structure, using the technique previously described in this section. In most cases, the shell script is not able to alter the values of the fields within these structures. The exception to this is the *XmTextVerifyCallbackStruct*, available during the *losingFocusCallback*, the *modifyVerifyCallback* and the *motionVerifyCallback* for the text widget. The *dtksh* utility supports the modification of certain fields within this structure, to the extent that it is supported by Motif. The following fields within the callback structure can be modified:

CB_CALL_DATA.DOIT
CB_CALL_DATA.STARTPOS
CB_CALL_DATA.ENDPOS
CB_CALL_DATA.TEXT.PTR
CB_CALL_DATA.TEXT.LENGTH
CB_CALL_DATA.TEXT.FORMAT

An example of how these fields can be modified:

```
CB_CALL_DATA.DOIT="false"
CB_CALL_DATA.TEXT.PTR="*"
CB_CALL_DATA.TEXT.LENGTH=1
```

Event Handler Context Variables

As with callbacks, an application registers event handlers with a widget to specify what action should occur when one of the specified events occurs. Again, the action can be any arbitrary *dtksh* command line. For example:

```
XtAddEventHandler $W "Button2MotionMask" false "ActivateProc"
XtAddEventHandler $W "ButtonPressMask|ButtonReleaseMask" \
       false "echo action"
```

Just as with callbacks, two environment variables are defined to provide context to the event handler:

EH_WIDGET
: Set to the widget handle for the widget for which the event handler is registered.

EH_EVENT
: Set to the address of the **XEvent** that triggered the event handler.

Access to the fields within the **XEvent** structure is the same as for the *CB_CALL_DATA* environment variable previously described in this section. For example:

```
if [${EH_EVENT.TYPE} = "ButtonPress"]; then
       echo X = ${EH_EVENT.XBUTTON.X}
       echo Y = ${EH_EVENT.XBUTTON.Y}
elif [${EH_EVENT.TYPE} = "KeyPress"]; then
       echo X = ${EH_EVENT.XKEY.X}
       echo Y = ${EH_EVENT.XKEY.Y}
fi
```

Translation Context Variables

Xt provides for event translations to be registered for a widget; their context is provided in the same way as with event handlers. The two variables defined for translation commands are:

TRANSLATION_WIDGET
: Set to the widget handle for the widget for which the translation is registered.

TRANSLATION_EVENT
: Set to the address of the **XEvent** that triggered the translation.

Dot-notation provides access to the fields of the event:

```
echo Event type = ${TRANSLATION_EVENT.TYPE}
echo Display = ${TRANSLATION_EVENT.XANY.DISPLAY}
```

Workspace Callback Context Variables

An application can register a callback function that is invoked any time the user changes to a new workspace. When the callback is invoked, the following two special environment variables are set, and can be accessed by the shell callback code:

CB_WIDGET
: Set to the widget handle for the widget invoking the callback.

CB_CALL_DATA
: Set to the X atom that uniquely identifies the new workspace. This can be converted to its string representation using the *XmGetAtomName* command.

Accessing Event Subfields

The **XEvent** structure has many different configurations based on the event's type. The *dtksh* utility provides access only to the most frequently used **XEvent**s. Any of the other standard **XEvent**s are accessed using the event type XANY, followed by any of the subfields defined by the **XANY** event structure, which includes the following subfields:

```
${TRANSLATION_EVENT.XANY.TYPE}
${TRANSLATION_EVENT.XANY.SERIAL}
${TRANSLATION_EVENT.XANY.SEND_EVENT}
${TRANSLATION_EVENT.XANY.DISPLAY}
${TRANSLATION_EVENT.XANY.WINDOW}
```

The *dtksh* utility supports full access to all of the event fields for the following event types:

```
XANY
XBUTTON
XEXPOSE
XNOEXPOSE
XGRAPHICSEXPOSE
XKEY
XMOTION
```

The following examples show how the subfields for the previously listed event types are accessed:

```
${TRANSLATION_EVENT.XBUTTON.X}
$(CB_CALL_DATA.EVENT.XKEY.STATE}
${EH_EVENT.XGRAPHICSEXPOSE.WIDTH}
```

Input Context Variables

Xt provides the *XtAddInput*( ) facility that allows an application to register interest in activity on a particular file descriptor. This generally includes data available for reading, the file descriptor being ready for writing, and exceptions on the file descriptor. If programming in C, the application provides a handler function that is invoked when the activity occurs. When reading data from the file descriptor, it is up to the handler to read the data from the input source and handle character escaping and line continuations.

The *dtksh* utility also supports the *XtAddInput*( ) facility, but has limited its functionality to reading data, and has taken the reading function a step further to make it easier for shell programmers to use. By default, when a shell script registers interest in a file descriptor, *dtksh* invokes the shell script's input handler only when a complete line of text has been received. A complete line of text is defined to be a line terminated either by an unescaped <newline> character, or by end-of-file. The input handler is also called if no data is available and end-of-file is reached. This gives the handler the opportunity to use *XtRemoveInput*( ) to remove the input source, and to close the file descriptor.

The advantage of this default behaviour is that input handlers do not need to do escape processing or handle line continuations. The disadvantage is that it assumes that all of the input is line-oriented and contains no binary information. If the input source does contain binary information, or if the input handler wants to read the data from the input source directly, *dtksh* also supports a raw input mode. In raw mode, *dtksh* does not read any of the data from the input source. Any time *dtksh* is notified that input is available on the input source, it invokes the shell script's input handler. It then becomes the handler's responsibility to read the incoming data, to perform any required buffering and escape processing, and to detect when end-of-file is

reached (so that the input source can be removed and the file descriptor closed).

Whether the input handler is configured to operate in the default mode or in raw mode, *dtksh* sets up several environment variables before calling the shell script's input handler. These environment variables provide the input handler with everything needed to handle the incoming data:

INPUT_LINE
: If operating in the default mode, this variable contains the next complete line of input available from the input source. If *INPUT_EOF* is set to True, there is no data in this buffer. If operating in raw mode, this environment variable always contains an empty string.

INPUT_EOF
: If operating in the default mode, this variable is set to False any time *INPUT_LINE* contains data, and is set to True when end-of-file is reached. When end-of-file is reached, the input handler for the shell script should unregister the input source and close the file descriptor. If operating in raw mode, *INPUT_EOF* is always set to False.

INPUT_SOURCE
: Indicates the file descriptor for which input is available. If operating in raw mode, this file descriptor is used to obtain the pending input. The file descriptor is also used to close the input source when it is no longer needed.

INPUT_ID
: Indicates the ID returned by *XtAddInput* when the input source was originally registered. This information is needed in order to remove the input source using *XtRemoveInput*.

ASYNCHRONOUS EVENTS

Default.

STDOUT

See *sh* in the X/Open CAE Specification, **Commands and Utilities**, **Issue 4, Version 2**.

STDERR

See *sh* in the X/Open CAE Specification, **Commands and Utilities**, **Issue 4, Version 2**.

OUTPUT FILES

None.

EXTENDED DESCRIPTION

The capabilities described here are extensions to those of the *sh* command language interpreter. See *sh* in the X/Open CAE Specification, **Commands and Utilities**, **Issue 4, Version 2**. The following subsections give a synopsis of each of the built-in commands added by *dtksh* to *sh*. In general, argument ordering and types are the same as for corresponding C procedures, with exceptions noted. For more detail on the functionality and arguments of a command, see the standard documentation for the corresponding X11, Xt, Motif or Desktop Services procedure.

In definitions listed in this document, arguments named *variable*, *variable2*, *variable3* and so on, indicate that the shell script must supply the name of an environment variable, into which some value is returned.

All of the Xt commands used to create a new widget require that the widget class for the new widget be specified. The widget (or gadget) class name is the standard class name provided by Motif. For example, the class name for a Motif pushbutton widget is *XmPushButton*, while the class name for the Motif label gadget is *XmLabelGadget*. Commands that use their exit status to

return a Boolean value (which can be used directly as part of an *if* statement) are noted as such.

Arguments enclosed within [] are optional.

Dtksh Built-in Xlib Commands

`XBell` *display volume*

`XClearArea` *display drawable* **[***optional GC arguments***]** *x y width height exposures*

`XClearWindow` *display drawable*

`XCopyArea` *display src dest srcX srcY width height destX destY* **[***optional GC arguments***]**

`XDefineCursor` *display window cursor*

`XDrawArc` *display drawable* **[***optional GC arguments***]** *x y width height angle1 angle2*

`XDrawLine` *display drawable* **[***optional GC arguments***]** *x1 y1 x2 y2*

`XDrawLines` *display drawable* **[***–coordinateMode***]** **[***optional GC arguments***]** *x1 y1 x2 y2* **[***x3 y3 ...***]**

The *coordinateMode* operand is either **CoordModeOrigin** or **CoordModePrevious**.

`XDrawPoint` *display drawable* **[***optional GC arguments***]** *x y*

`XDrawPoints` *display drawable* **[***–coordinateMode***]** **[***optional GC arguments***]** *x1 y1* **[***x2 y2 x3 y3 ...***]**

The *coordinateMode* operand is either **CoordModeOrigin** or **CoordModePrevious**.

`XDrawRectangle` *display drawable* **[***optional GC arguments***]** *x y width height*

`XDrawSegments` *display drawable* **[***optional GC arguments***]** *x1 y1 x2 y2* **[***x3 y3 x4 y4 ...***]**

`XDrawString` *display drawable* **[***optional GC arguments***]** *x y string*

`XDrawImageString` *display drawable* **[***optional GC arguments***]** *x y string*

`XFillArc` *display drawable* **[***optional GC arguments***]** *x y width height angle1 angle2*

`XFillPolygon` *display drawable* **[***–shape***]** **[***–coordinateMode***]** **[***optional GC arguments***]** *x1 y1 x2 y2 ...*

The *shape* operand is one of **Complex**, **Convex** or **Nonconvex**, and where *coordinateMode* is either **CoordModeOrigin** or **CoordModePrevious**.

`XFillRectangle` *display drawable* **[***optional GC arguments***]** *x y width height*

`XFlush` *display*

`XHeightOfScreen` *variable screen*

`XRaiseWindow` *display window*

`XRootWindowOfScreen` *variable screen*

`XSync` *`display discard`*

The *discard* operand is either True or False.

`XTextWidth` *`variable fontName string`*

The *XTextWidth* command differs from the C procedure; it takes the name of a font instead of a pointer to a font structure.

`XUndefineCursor` *`display window`*

`XWidthOfScreen` *`variable screen`*

Built-in XtIntrinsic Commands

`XtAddCallback` *`widgetHandle callbackName dtksh-command`*

The *callbackName* operand is one of the standard Motif or Xt callback names, with the Xt or Xm prefix omitted; for example, *activateCallback*.

`XtAddEventHandler` *`widgetHandle eventMask nonMaskableFlag dtksh-command`*

The *eventMask* operand is of the form *mask* | *mask* | *mask* and the *mask* component is any of the standard set of **XEvent** masks; for example, *ButtonPressMask*, where *nonMaskableFlag* is either True or False.

`XtAddInput` *`variable`* **`[`**`-r`**`]`** *`fileDescriptor dtksh-command`*

The *XtAddInput* command registers the indicated file descriptor with the X Toolkit as an alternative input source (that is, for reading). The input handler for the shell script is responsible for unregistering the input source when it is no longer needed, and also to close the file descriptor. If the **–r** option is specified (raw mode), *dtksh* does not automatically read any of the data available from the input source; it is up to the specified *dtksh* command to read all data. If the **–r** option is not specified, the specified *dtksh* command is invoked only when a full line has been read (that is, a line terminated by either an unescaped <newline> character, or end-of-file) and when end-of-file is reached. The raw mode is useful for handlers expecting to process non-textual data, or for handlers not wanting *dtksh* to automatically read in a line of data. When end-of-file is detected, it is the responsibility of the input handler for the shell script to use *XtRemoveInput* to remove the input source, and to close the file descriptor, if necessary. In all cases, several environment variables are set up for the handler to use. These include the following:

INPUT_LINE
Empty if raw mode; otherwise, contains next line to be processed.

INPUT_EOF
Set to True if end-of-file reached; otherwise, set to False.

INPUT_SOURCE
File descriptor associated with this input source.

INPUT_ID
ID associated with this input handler; returned by *XtAddInput*.

`XtAddTimeout` *`variable interval dtksh-command`*

`XtAddWorkProc` *`variable dtksh-command`*

In *dtksh*, the *dtksh-command* is typically a *dtksh* function name. Like regular work procedures, this function is expected to return a value indicating whether the work

procedure wants to be called again, or whether it has completed its work and can be automatically unregistered. If the *dtksh* function returns zero, the work procedure remains registered; any other value causes the work procedure to be automatically unregistered.

`XtAugmentTranslations` *widgetHandle translations*

`XtCreateApplicationShell` *variable applicationName widgetClass* **[***resource:value ...***]**

`XtCallCallbacks` *widgetHandle callbackName*

The *callbackName* operand is one of the standard Motif or Xt callback names, with the Xt or Xm prefix omitted; for example, *activateCallback*.

`XtClass` *variable widgetHandle*

The command returns the name of the widget class associated with the passed-in widget handle.

`XtCreateManagedWidget` *variable widgetName widgetClass parentWidgetHandle* **[***resource:value ...***]**

`XtCreatePopupShell` *variable widgetName widgetClass parentWidgetHandle* **[***resource:value ...***]**

`XtCreateWidget` *variable widgetName widgetClass parentWidgetHandle* **[***resource:value ...***]**

`XtDestroyWidget` *widgetHandle* **[***widgetHandle ...***]**

`XtDisplay` *variable widgetHandle*

`XtDisplayOfObject` *variable widgetHandle*

`XtGetValues` *widgetHandle resource:variable1* **[***resource:variable2 ...***]**

`XtHasCallbacks` *variable widgetHandle callbackName*

The *callbackName* operand is one of the standard Motif or Xt callback names, with the Xt or Xm prefix omitted: for example, *activateCallback* variable is set to one of the strings **CallbackNoList**, **CallbackHasNone** or **CallbackHasSome**.

`XtInitialize` *variable shellName applicationClassName applicationName arguments*

Similar to a typical Motif-based program, the *arguments* argument is used to reference any command-line arguments that might have been specified by the shell script user; these are typically referred using the shell syntax of *$@*. The *applicationName* argument is listed because *$@* does not include *$0*. The *applicationName* and *arguments* are used to build the argument list passed to the *XtInitialize* command. Upon completion, the environment variable *DTKSH_ARGV* is set to the argument list as returned by the *XtInitialize* command; the *DTKSH_TOPLEVEL* environment variable is set to the widget handle of the widget created by *XtInitialize*, and the *DTKSH_APPNAME* environment variable is set to the value of the *applicationName* argument. The command returns a value that can be used in a conditional.

`XtIsManaged` *widgetHandle*

The command returns a value that can be used in a conditional.

`XtIsSubclass` *`widgetHandle widgetClass`*

The *widgetClass* operand is the name of a widget class. The command returns a value that can be used in a conditional.

`XtNameToWidget` *`variable referenceWidget name`*

`XtIsRealized` *`widgetHandle`*

The command returns a value that can be used in a conditional.

`XtIsSensitive` *`widgetHandle`*

The command returns a value that can be used in a conditional.

`XtIsShell` *`widgetHandle`*

The command returns a value that can be used in a conditional.

`XtLastTimestampProcessed` *`variable display`*

`XtMainLoop`

`XtManageChild` *`widgetHandle`*

`XtManageChildren` *`widgetHandle`* **[***`widgetHandle ...`***]**

`XtMapWidget` *`widgetHandle`*

`XtOverrideTranslations` *`widgetHandle translations`*

`XtParent` *`variable widgetHandle`*

`XtPopdown` *`widgetHandle`*

`XtPopup` *`widgetHandle grabType`*

The *grabType* operand is one of the strings **GrabNone**, **GrabNonexclusive** or **GrabExclusive**.

`XtRealizeWidget` *`widgetHandle`*

`XtRemoveAllCallbacks` *`widgetHandle callbackName`*

The *callbackName* operand is one of the standard Motif or Xt callback names, with the Xt or Xm prefix omitted; for example, *activateCallback*.

`XtRemoveCallback` *`widgetHandle callbackName dtksh-command`*

The *callbackName* operand is one of the standard Motif or Xt callback names, with the Xt or Xm prefix omitted; for example, *activateCallback*. As with traditional Xt callbacks, when a callback is removed, the same *dtksh* command string must be specified as was specified when the callback was originally registered.

`XtRemoveEventHandler` *`widgetHandle eventMask nonMaskableFlag dtksh-command`*

The *eventMask* operand is of the form *mask* | *mask* | *mask* and the *mask* component is any of the standard set of **XEvent** masks; for example, *ButtonPressMask*, where *nonMaskableFlag* is either True or False. As with traditional Xt event handlers, when an event handler is removed, the same *eventMask*, *nonMaskableFlag* setting and *dtksh* command string must be specified as was specified when the event handler was originally registered.

`XtRemoveInput` *`inputId`*

The *inputId* operand is the handle returned in the specified environment variable when the alternative input source was registered using the *XtAddInput* command.

`XtRemoveTimeOut` *`timeoutId`*

The *timeoutId* operand is the handle returned in the specified environment variable when the timeout was registered using the *XtAddTimeOut* command.

`XtRemoveWorkProc` *`workprocId`*

The *workprocId* operand is the handle returned in the specified environment variable when the work procedure was registered using the *XtAddWorkProc* command.

`XtScreen` *`variable widgetHandle`*

`XtSetSensitive` *`widgetHandle state`*

The *state* operand is either True or False.

`XtSetValues` *`widgetHandle resource:value`* **[***`resource:value ...`***]**

`XtUninstallTranslations` *`widgetHandle`*

`XtUnmanageChild` *`widgetHandle`*

`XtUnmanageChildren` *`widgetHandle`* **[***`widgetHandle ...`***]**

`XtUnmapWidget` *`widgetHandle`*

`XtUnrealizeWidget` *`widgetHandle`*

`XtWindow` *`variable widgetHandle`*

Built-in Motif Commands

`XmAddWMProtocolCallback` *`widgetHandle protocolAtom dtksh-command`*

The *protocolAtom* operand is typically obtained using the *XmInternAtom* command.

`XmAddWMProtocols` *`widgetHandle protocolAtom`* **[***`protocolAtom ...`***]**

The *protocolAtom* operand is typically obtained using the *XmInternAtom* command.

`XmCommandAppendValue` *`widgetHandle string XmCommandError widgetHandle errorString`*

`XmCommandGetChild` *`variable widgetHandle childType`*

The *childType* operand is one of the strings:

```
DIALOG_COMMAND_TEXT
DIALOG_PROMPT_LABEL
DIALOG_HISTORY_LIST
DIALOG_WORK_AREA
```

`XmCommandSetValue` *`widgetHandle commandString`*

`XmCreateArrowButton` *`variable parentWidgetHandle name`* **[***`resource:value ...`***]**

`XmCreateArrowButtonGadget` *`variable parentWidgetHandle name`* **[***`resource:value ...`***]**

XmCreateBulletinBoard *variable parentWidgetHandle name*
[*resource:value ...***]**

XmCreateBulletinBoardDialog *variable parentWidgetHandle name*
[*resource:value ...***]**

XmCreateCascadeButton *variable parentWidgetHandle name*
[*resource:value ...***]**

XmCreateCascadeButtonGadget *variable parentWidgetHandle name*
[*resource:value ...***]**

XmCreateCommand *variable parentWidgetHandle name* **[***resource:value ...***]**

XmCreateDialogShell *variable parentWidgetHandle name*
[*resource:value ...***]**

XmCreateDrawingArea *variable parentWidgetHandle name*
[*resource:value ...***]**

XmCreateDrawnButton *variable parentWidgetHandle name*
[*resource:value ...***]**

XmCreateErrorDialog *variable parentWidgetHandle name*
[*resource:value ...***]**

XmCreateFileSelectionBox *variable parentWidgetHandle name*
[*resource:value ...***]**

XmCreateFileSelectionDialog *variable parentWidgetHandle name*
[*resource:value ...***]**

XmCreateForm *variable parentWidgetHandle name* **[***resource:value ...***]**

XmCreateFormDialog *variable parentWidgetHandle name*
[*resource:value ...***]**

XmCreateFrame *variable parentWidgetHandle name* **[***resource:value ...***]**

XmCreateInformationDialog *variable parentWidgetHandle name*
[*resource:value ...***]**

XmCreateLabel *variable parentWidgetHandle name* **[***resource:value ...***]**

XmCreateLabelGadget *variable parentWidgetHandle name*
[*resource:value ...***]**

XmCreateList *variable parentWidgetHandle name* **[***resource:value ...***]**

XmCreateMainWindow *variable parentWidgetHandle name*
[*resource:value ...***]**

XmCreateMenuBar *variable parentWidgetHandle name* **[***resource:value ...***]**

XmCreateMenuShell *variable parentWidgetHandle name* **[***resource:value ...***]**

XmCreateMessageBox *variable parentWidgetHandle name*
[*resource:value ...***]**

XmCreateMessageDialog *variable parentWidgetHandle name*
[*resource:value ...***]**

XmCreateOptionMenu *variable parentWidgetHandle name*
[*resource:value ...***]**

XmCreatePanedWindow *variable parentWidgetHandle name*
[*resource:value ...***]**

XmCreatePopupMenu *variable parentWidgetHandle name* **[***resource:value ...***]**

XmCreatePromptDialog *variable parentWidgetHandle name*
[*resource:value ...***]**

XmCreatePulldownMenu *variable parentWidgetHandle name*
[*resource:value ...***]**

XmCreatePushButton *variable parentWidgetHandle name*
[*resource:value ...***]**

XmCreatePushButtonGadget *variable parentWidgetHandle name*
[*resource:value ...***]**

XmCreateQuestionDialog *variable parentWidgetHandle name*
[*resource:value ...***]**

XmCreateRadioBox *variable parentWidgetHandle name* **[***resource:value ...***]**

XmCreateRowColumn *variable parentWidgetHandle name* **[***resource:value ...***]**

XmCreateScale *variable parentWidgetHandle name* **[***resource:value ...***]**

XmCreateScrollBar *variable parentWidgetHandle name* **[***resource:value ...***]**

XmCreateScrolledList *variable parentWidgetHandle name*
[*resource:value ...***]**

XmCreateScrolledText *variable parentWidgetHandle name*
[*resource:value ...***]**

XmCreateScrolledWindow *variable parentWidgetHandle name*
[*resource:value ...***]**

XmCreateSelectionBox *variable parentWidgetHandle name*
[*resource:value ...***]**

XmCreateSelectionDialog *variable parentWidgetHandle name*
[*resource:value ...***]**

XmCreateSeparator *variable parentWidgetHandle name* **[***resource:value ...***]**

XmCreateSeparatorGadget *variable parentWidgetHandle name*
[*resource:value ...***]**

XmCreateText *variable parentWidgetHandle name* **[***resource:value ...***]**

XmCreateTextField *variable parentWidgetHandle name* **[***resource:value ...***]**

XmCreateToggleButton *variable parentWidgetHandle name*
[*resource:value ...***]**

XmCreateToggleButtonGadget *variable parentWidgetHandle name*
[*resource:value ...***]**

XmCreateWarningDialog *variable parentWidgetHandle name*
[*resource:value ...***]**

XmCreateWorkArea *variable parentWidgetHandle name* **[***resource:value ...***]**

`XmCreateWorkingDialog` *variable parentWidgetHandle name* **[***resource:value ...***]**

`XmFileSelectionDoSearch` *widgetHandle directoryMask*

`XmFileSelectionBoxGetChild` *variable widgetHandle childType*

The *childType* operand is one of the strings:

```
DIALOG_APPLY_BUTTON
DIALOG_CANCEL_BUTTON
DIALOG_DEFAULT_BUTTON
DIALOG_DIR_LIST
DIALOG_DIR_LIST_LABEL
DIALOG_FILTER_LABEL
DIALOG_FILTER_TEXT
DIALOG_HELP_BUTTON
DIALOG_LIST
DIALOG_LIST_LABEL
DIALOG_OK_BUTTON
DIALOG_SEPARATOR
DIALOG_SELECTION_LABEL
DIALOG_TEXT
DIALOG_WORK_AREA
```

`XmGetAtomName` *variable display atom*

`XmGetColors` *widgetHandle background variable variable2 variable3 variable4*

The *XmGetColors* command differs from the C procedure in that it takes a *widgetHandle* instead of a screen pointer and a colourmap.

`XmGetFocusWidget` *variable widgetHandle*

`XmGetPostedFromWidget` *variable widgetHandle*

`XmGetTabGroup` *variable widgetHandle*

`XmGetTearOffControl` *variable widgetHandle*

`XmGetVisibility` *variable widgetHandle*

`XmInternAtom` *variable display atomString onlyIfExistsFlag*

The *onlyIfExistsFlag* operand can be set to either True or False.

`XmIsTraversable` *widgetHandle*

The command returns a value that can be used in a conditional.

`XmListAddItem` *widgetHandle position itemString*

The ordering of the arguments to the *XmListAddItem* command differs from the corresponding C function.

`XmListAddItems` *widgetHandle position itemString* **[***itemString ...***]**

The ordering of the arguments to the *XmListAddItems* command differs from the corresponding C function.

`XmListAddItemsUnselected` *`widgetHandle position itemString`* **[***`itemString ...`***]**

The ordering of the arguments to the *XmListAddItemsUnselected* command differs from the corresponding C function.

`XmListAddItemUnselected` *`widgetHandle position itemString`*

The ordering of the arguments to the *XmListAddItemUnselected* command differs from the corresponding C function.

`XmListDeleteAllItems` *`widgetHandle`*

`XmListDeleteItem` *`widgetHandle itemString`*

`XmListDeleteItems` *`widgetHandle itemString`* **[***`itemString ...`***]**

`XmListDeleteItemsPos` *`widgetHandle itemCount position`*

`XmListDeletePos` *`widgetHandle position`*

`XmListDeletePositions` *`widgetHandle position`* **[***`position ...`***]**

`XmListDeselectAllItems` *`widgetHandle`*

`XmListDeselectItem` *`widgetHandle itemString`*

`XmListDeselectPos` *`widgetHandle position`*

`XmListGetSelectedPos` *`variable widgetHandle`*

The command returns in *variable* a comma-separated list of indices. The command returns a value that can be used in a conditional.

`XmListGetKbdItemPos` *`variable widgetHandle`*

`XmListGetMatchPos` *`variable widgetHandle itemString`*

The command returns in *variable* a comma-separated list of indices. The command returns a value that can be used in a conditional.

`XmListItemExists` *`widgetHandle itemString`*

The command returns a value that can be used in a conditional.

`XmListItemPos` *`variable widgetHandle itemString`*

`XmListPosSelected` *`widgetHandle position`*

The command returns a value that can be used in a conditional.

`XmListPosToBounds` *`widgetHandle position variable variable2 variable3 variable4`*

The command returns a value that can be used in a conditional.

`XmListReplaceItemsPos` *`widgetHandle position itemString`* **[***`itemString ...`***]**

The ordering of the arguments to the *XmListReplaceItemsPos* command differs from the corresponding C function.

`XmListReplaceItemsPosUnselected` *`widgetHandle position itemString`* **[***`itemString ...`***]**

The ordering of the arguments to the *XmListReplaceItemsPosUnselected* command differs from the corresponding C function.

`XmListSelectItem` *`widgetHandle itemString notifyFlag`*

The *notifyFlag* operand can be set to either True or False.

`XmListSelectPos` *`widgetHandle position notifyFlag`*

The *notifyFlag* operand can be set to either True or False.

`XmListSetAddMode` *`widgetHandle state`*

The *state* operand can be set to either True or False.

`XmListSetBottomItem` *`widgetHandle itemString`*

`XmListSetBottomPos` *`widgetHandle position`*

`XmListSetHorizPos` *`widgetHandle position`*

`XmListSetItem` *`widgetHandle itemString`*

`XmListSetKbdItemPos` *`widgetHandle position`*

The command returns a value that can be used in a conditional.

`XmListSetPos` *`widgetHandle position`*

`XmListUpdateSelectedList` *`widgetHandle`*

`XmMainWindowSep1` *`variable widgetHandle`*

`XmMainWindowSep2` *`variable widgetHandle`*

`XmMainWindowSep3` *`variable widgetHandle`*

`XmMainWindowSetAreas` *`widgetHandle menuWidgetHandle commandWidgetHandle horizontalScrollbarWidgetHandle verticalScrollbarWidgetHandle workRegionWidgetHandle`*

`XmMenuPosition` *`widgetHandle eventHandle`*

The *eventHandle* operand refers to an **XEvent** that has typically been obtained by accessing the *CB_CALL_DATA.EVENT*, *EH_EVENT* or *TRANSLATION_EVENT* environment variables.

`XmMessageBoxGetChild` *`variable widgetHandle childType`*

The *childType* operand is one of the strings:

```
DIALOG_CANCEL_BUTTON
DIALOG_DEFAULT_BUTTON
DIALOG_HELP_BUTTON
DIALOG_MESSAGE_LABEL
DIALOG_OK_BUTTON
DIALOG_SEPARATOR
DIALOG_SYMBOL_LABEL
```

`XmOptionButtonGadget` *`variable widgetHandle`*

`XmOptionLabelGadget` *`variable widgetHandle`*

`XmProcessTraversal` *`widgetHandle direction`*

The *direction* operand is one of the strings:

```
TRAVERSE_CURRENT
TRAVERSE_DOWN
TRAVERSE_HOME
TRAVERSE_LEFT
TRAVERSE_NEXT
TRAVERSE_NEXT_TAB_GROUP
TRAVERSE_PREV
TRAVERSE_PREV_TAB_GROUP
TRAVERSE_RIGHT
TRAVERSE_UP
```

The command returns a value that can be used in a conditional.

`XmRemoveWMProtocolCallback` *`widgetHandle protocolAtom dtksh-command`*

The *protocolAtom* operand is typically obtained using the *XmInternAtom* command. As with traditional WM callbacks, when a callback is removed, the same *dtksh* command string must be specified as was specified when the callback was originally registered.

`XmRemoveWMProtocols` *`widgetHandle protocolAtom`* **[***`protocolAtom ...`***]**

The *protocolAtom* operand is typically obtained using the *XmInternAtom* command.

`XmScaleGetValue` *`widgetHandle variable`*

`XmScaleSetValue` *`widgetHandle value`*

`XmScrollBarGetValues` *`widgetHandle variable variable2 variable3 variable4`*

`XmScrollBarSetValues` *`widgetHandle value sliderSize increment pageIncrement notifyFlag`*

The *notifyFlag* operand can be set to either True or False.

`XmScrollVisible` *`widgetHandle widgetHandle leftRightMargin topBottomMargin`*

`XmSelectionBoxGetChild` *`variable widgetHandle childType`*

The *childType* operand is one of the strings:

```
DIALOG_CANCEL_BUTTON
DIALOG_DEFAULT_BUTTON
DIALOG_HELP_BUTTON
DIALOG_APPLY_BUTTON
DIALOG_LIST
DIALOG_LIST_LABEL
DIALOG_OK_BUTTON
DIALOG_SELECTION_LABEL
DIALOG_SEPARATOR
DIALOG_TEXT
DIALOG_WORK_AREA
```

`XmTextClearSelection` *`widgetHandle time`*

The *time* operand is typically either obtained from within an **XEvent**, or from a call to the *XtLastTimestampProcessed* command.

`XmTextCopy` *`widgetHandle time`*

The *time* operand is typically either obtained from within an **XEvent**, or from a call to the *XtLastTimestampProcessed* command. The command returns a value that can be used in a conditional.

`XmTextCut` *`widgetHandle time`*

The *time* operand is typically either obtained from within an **XEvent**, or from a call to the *XtLastTimestampProcessed* command. The command returns a value that can be used in a conditional.

`XmTextDisableRedisplay` *`widgetHandle`*

`XmTextEnableDisplay` *`widgetHandle`*

`XmTextFindString` *`widgetHandle startPosition string direction variable`*

The *direction* operand is one of the strings TEXT_FORWARD or TEXT_BACKWARD. The command returns a value that can be used in a conditional.

`XmTextGetBaseline` *`variable widgetHandle`*

`XmTextGetEditable` *`widgetHandle`*

The command returns a value that can be used in a conditional.

`XmTextGetInsertionPosition` *`variable widgetHandle`*

`XmTextGetLastPosition` *`variable widgetHandle`*

`XmTextGetMaxLength` *`variable widgetHandle`*

`XmTextGetSelection` *`variable widgetHandle`*

`XmTextGetSelectionPosition` *`widgetHandle variable variable2`*

The command returns a value that can be used in a conditional.

`XmTextGetString` *`variable widgetHandle`*

`XmTextGetTopCharacter` *`variable widgetHandle`*

`XmTextInsert` *`widgetHandle position string`*

`XmTextPaste` *`widgetHandle`*

The command returns a value that can be used in a conditional.

`XmTextPosToXY` *`widgetHandle position variable variable2`*

The command returns a value that can be used in a conditional.

`XmTextRemove` *`widgetHandle`*

The command returns a value that can be used in a conditional.

`XmTextReplace` *`widgetHandle fromPosition toPosition string`*

`XmTextScroll` *`widgetHandle lines`*

`XmTextSetAddMode` *`widgetHandle state`*

The *state* operand can be set to either True or False.

`XmTextSetEditable` *`widgetHandle editableFlag`*

The *editableFlag* operand can be set to either True or False.

`XmTextSetHighlight` *`widgetHandle leftPosition rightPosition mode`*

The *mode* operand is one of the strings:

```
HIGHLIGHT_NORMAL
HIGHLIGHT_SELECTED
HIGHLIGHT_SECONDARY_SELECTED
```

`XmTextSetInsertionPosition` *`widgetHandle position`*

`XmTextSetMaxLength` *`widgetHandle maxLength`*

`XmTextSetSelection` *`widgetHandle firstPosition lastPosition time`*

The *time* operand is typically either obtained from within an **XEvent**, or from a call to the *XtLastTimestampProcessed* command.

`XmTextSetString` *`widgetHandle string`*

`XmTextSetTopCharacter` *`widgetHandle topCharacterPosition`*

`XmTextShowPosition` *`widgetHandle position`*

`XmTextXYToPos` *`variable widgetHandle x y`*

`XmTextFieldClearSelection` *`widgetHandle time`*

The *time* operand is typically either obtained from within an **XEvent**, or from a call to the *XtLastTimestampProcessed* command.

`XmTextFieldGetBaseline` *`variable widgetHandle`*

`XmTextFieldGetEditable` *`widgetHandle`*

The command returns a value that can be used in a conditional.

`XmTextFieldGetInsertionPosition` *`variable widgetHandle`*

`XmTextFieldGetLastPosition` *`variable widgetHandle`*

`XmTextFieldGetMaxLength` *`variable widgetHandle`*

`XmTextFieldGetSelection` *`variable widgetHandle`*

`XmTextFieldGetSelectionPosition` *`widgetHandle variable variable2`*

The command returns a value that can be used in a conditional.

`XmTextFieldGetString` *`variable widgetHandle`*

`XmTextFieldInsert` *`widgetHandle position string`*

`XmTextFieldPosToXY` *`widgetHandle position variable variable2`*

The command returns a value that can be used in a conditional.

`XmTextFieldRemove` *`widgetHandle`*

The command returns a value that can be used in a conditional.

`XmTextFieldReplace` *`widgetHandle fromPosition toPosition string`*

`XmTextFieldSetEditable` *`widgetHandle editableFlag`*

The *editableFlag* operand can be set to either True or False.

`XmTextFieldSetHighlight` *`widgetHandle leftPosition rightPosition mode`*

The *mode* operand is one of the strings:

```
HIGHLIGHT_NORMAL
HIGHLIGHT_SELECTED
HIGHLIGHT_SECONDARY_SELECTED
```

`XmTextFieldSetInsertionPosition` *`widgetHandle position`*

`XmTextFieldSetMaxLength` *`widgetHandle maxLength`*

`XmTextFieldSetSelection` *`widgetHandle firstPosition lastPosition time`*

The *time* operand is typically either obtained from within an **XEvent**, or from a call to the *XtLastTimestampProcessed* command.

`XmTextFieldSetString` *`widgetHandle string`*

`XmTextFieldShowPosition` *`widgetHandle position`*

`XmTextFieldXYToPos` *`variable widgetHandle x y`*

`XmTextFieldCopy` *`widgetHandle time`*

The *time* operand is typically either obtained from within an **XEvent**, or from a call to the *XtLastTimestampProcessed* command. The command returns a value that can be used in a conditional.

`XmTextFieldCut` *`widgetHandle time`*

The *time* operand is typically either obtained from within an **XEvent** or from a call to the *XtLastTimestampProcessed* command. The command returns a value that can be used in a conditional.

`XmTextFieldPaste` *`widgetHandle`*

The command returns a value that can be used in a conditional.

`XmTextFieldSetAddMode` *`widgetHandle state`*

The *state* operand can be set to either True or False.

`XmToggleButtonGadgetGetState` *`widgetHandle`*

The command returns a value that can be used in a conditional.

`XmToggleButtonGadgetSetState` *`widgetHandle state notifyFlag`*

The *state* operand can be set to either True or False. The *notifyFlag* operand can be set to either True or False.

`XmToggleButtonGetState` *`widgetHandle`*

The command returns a value that can be used in a conditional.

`XmToggleButtonSetState` *`widgetHandle state notifyFlag`*

The *state* operand can be set to either True or False. The *notifyFlag* operand can be set to either True or False.

`XmUpdateDisplay` *`widgetHandle`*

Built-in XCDE Application Help Commands

`DtCreateHelpQuickDialog` *`variable parentWidgetHandle name`* **[***`resource:value ...`***]**

`DtCreateHelpDialog` *`variable parentWidgetHandle name`* **[***`resource:value ...`***]**

`DtHelpQuickDialogGetChild` *`variable widgetHandle childType`*

The *childType* operand is one of the strings:

```
HELP_QUICK_OK_BUTTON
HELP_QUICK_PRINT_BUTTON
HELP_QUICK_HELP_BUTTON
HELP_QUICK_SEPARATOR
HELP_QUICK_MORE_BUTTON
HELP_QUICK_BACK_BUTTON
```

`DtHelpReturnSelectedWidgetId` *`variable widgetHandle variable2`*

The *variable* operand is set to one of the strings:

```
HELP_SELECT_VALID
HELP_SELECT_INVALID
HELP_SELECT_ABORT
HELP_SELECT_ERROR
```

and *variable2* is set to the *widgetHandle* for the selected widget.

`DtHelpSetCatalogName` *`catalogName`*

Built-in Localisation Commands

`catopen` *`variable catalogName`*

Opens the indicated message catalogue, and returns the catalogue ID in the environment variable specified by *variable*. If a shell script needs to close the file descriptor associated with a message catalogue, the catalogue ID must be closed using the *catclose* command.

`catgets` *`variable catalogId setNumber messageNumber defaultMessageString`*

Attempts to extract the requested message string from the message catalogue associated with the *catalogId* argument. If the message string cannot be located, the default message string is returned. In either case, the returned message string is placed into the environment variable indicated by *variable*.

`catclose` *`catalogId`*

Closes the message catalogue associated with the indicated *catalogId*.

Built-in Session Management Commands

`DtSessionRestorePath` *`widgetHandle variable sessionFile`*

Given the filename for the session file (excluding any path information), this command returns the full pathname for the session file in the environment variable indicated by *variable*. The command returns a value that can be used in a conditional, indicating whether the command succeeded.

`DtSessionSavePath` *`widgetHandle variable variable2`*

The full pathname for the session file is returned in environment variable indicated by *variable*. The filename portion of the session file (excluding any path information) is returned in the environment variable indicated by *variable2*. The command returns a value that can be used in a conditional, indicating whether the command succeeded.

`DtShellIsIconified` *`widgetHandle`*

The command returns a value that can be used in a conditional.

`DtSetStartupCommand` *`widgetHandle commandString`*

Part of the session management process is telling the session manager how to restart the application the next time the user reopens the session. This command passes along the specified command string to the session manager. The widget handle should refer to an application shell.

`DtSetIconifyHint` *`widgetHandle iconifyHint`*

The *iconifyHint* operand can be set to either True or False. This command sets the initial iconified state for a shell window. This command only works if the window associated with the widget has not yet been realised.

Built-in Workspace Management Commands

`DtWsmAddCurrentWorkspaceCallback` *`variable widgetHandle dtksh-command`*

This command evaluates the specified *dtksh* command whenever the user changes workspaces. The handle associated with this callback is returned in the environment variable indicated by *variable*. The widget indicated by *widgetHandle* should be a shell widget.

`DtWsmRemoveWorkspaceCallback` *`callback-handle`*

The *callback-handle* must be a handle that was returned by *DtWsmAddCurrentWorkspaceCallback*.

`DtWsmGetCurrentWorkspace` *`display rootWindow variable`*

This command returns the X atom representing the user's current workspace in the environment variable indicated by *variable*. The *XmGetAtomName* command maps the X atom into its string representation.

`DtWsmSetCurrentWorkspace` *`widgetHandle workspaceNameAtom`*

This command changes the user's current workspace to the workspace indicated by *workspaceNameAtom*. The command returns a value that can be used in a conditional, indicating whether the command succeeded.

`DtWsmGetWorkspaceList` *`display rootWindow variable`*

This command returns in *variable* a string of comma-separated X atoms, representing the current set of workspaces defined for the user. The command returns a value that can be used in a conditional, indicating whether the command succeeded.

`DtWsmGetWorkspacesOccupied` *`display window variable`*

This command returns a string of comma-separated X atoms, representing the current set of workspaces occupied by the indicated shell window in the environment variable indicated by *variable*. The command returns a value that can be used in a conditional, indicating whether the command succeeded.

`DtWsmSetWorkspacesOccupied` *`display window workspaceList`*

This command moves the indicated shell window to the set of workspaces indicated by the string *workspaceList*, which must be a comma-separated list of X atoms.

`DtWsmAddWorkspaceFunctions` *`display window`*

`DtWsmRemoveWorkspaceFunctions` *`display window`*

`DtWsmOccupyAllWorkspaces` *`display window`*

`DtWsmGetCurrentBackdropWindows` *`display rootWindow variable`*

This command returns in *variable* a string of comma-separated window IDs representing the set of root backdrop windows.

Built-in Action Commands

The set of commands in this section provides the programmer with the tools for loading the action databases, querying information about actions defined in the databases, and requesting that an action be initiated.

`DtDbLoad`

This command reads in the action and data types databases. It must be called before any of the other Action or Data Typing Commands. The shell script should also use the *DtDbReloadNotify* command so that the shell script can be notified if new databases must be loaded.

`DtDbReloadNotify` *`dtksh-command`*

The specified *dtksh* command is executed when the notification is received. Typically, the *dtksh* command includes a call to the *DtDbLoad* command.

`DtActionExists` *`actionName`*

The command returns a value that can be used in a conditional.

`DtActionLabel` *`variable actionName`*

If the action does not exist, then an empty string is returned.

`DtActionDescription` *`variable actionName`*

This command returns an empty string if the action is not defined, or if the DESCRIPTION attribute is not specified.

`DtActionInvoke` *`widgetHandle actionName termOpts execHost contextDir useIndicator dtksh-command`* **[**`FILE` *`fileName`***]** `...`

The [FILE *fileName*] couplets can be used to specify file arguments to be used by *DtActionInvoke* when invoking the specified action. The *dtksh-command* argument must be specified as a null ("") value.

Built-in Data Typing Commands

`DtDtsLoadDataTypes`

This command should be invoked before any of the other data typing commands.

`DtDtsFileToDataType` *`variable filePath`*

This command returns the name of the data type associated with the file indicated by the *filePath* argument in the *variable* argument. The *variable* argument is set to an empty string if the file cannot be typed.

`DtDtsFileToAttributeValue` *`variable filePath attrName`*

This command returns the string representing the value of the specified attribute for the data type associated with the indicated file in the *variable* argument. If the attribute is not defined, or if the file cannot be typed, the *variable* argument is set to an empty string.

`DtDtsFileToAttributeList` *`variable filePath`*

This command returns the space-separated list of attribute names defined for the data type associated with the indicated file in the *variable* argument. A shell script queries the individual values for the attributes using the *DtDtsFileToAttributeValue* command. The *variable* argument is set to an empty string if the file cannot be typed. This command differs from the corresponding C function in that it only returns the names of the defined attributes and not their values.

`DtDtsDataTypeToAttributeValue` *`variable dataType attrName optName`*

This command returns the string representing the value of the specified attribute for the indicated data type in *variable*. If the attribute is not defined, or if the indicated data type does not exist, the *variable* argument is set to an empty string.

`DtDtsDataTypeToAttributeList` *`variable dataType optName`*

This command returns the space-separated list of attribute names defined for the indicated data type in *variable*. A shell script queries the individual values for the attributes using the *DtDtsDataTypeToAttributeValue* command. The *variable* argument is set to an empty string if the data type is not defined. This command differs from the corresponding C function in that it only returns the names of the defined attributes, and not their values.

`DtDtsFindAttribute` *`variable name value`*

This command returns a space-separated list of data type names whose attribute, indicated by the *name* argument, has the value indicated by the *value* argument. If an error occurs, the *variable* argument is set to an empty string.

`DtDtsDataTypeNames` *`variable`*

This command returns a space-separated list representing all of the data types currently defined in the data types database. If an error occurs, the *variable* argument is set to an empty string.

`DtDtsSetDataType` *`variable filePath dataType override`*

The *variable* argument is set to the resultant saved data type for the directory.

`DtDtsDataTypeIsAction` *`dataType`*

The command returns a value that can be used in a conditional.

Built-in XCDE Desktop Services Message Set Commands

The following set of commands implement a subset of the Desktop Services Message Set, allowing shell script participation in the Desktop Services protocol. Many of the ToolTalk commands differ slightly from their associated C programming call. For ToolTalk commands that typically return a pointer, a C application can validate that pointer by calling the *tt_ptr_error*( ) function; this C function call returns a **Tt_status** value, which indicates whether the pointer was valid, and if not, why it was not. In *dtksh*, all of the Desktop Services Message Set Commands that return a pointer also return the associated **Tt_status** value for the pointer automatically; this saves the shell script from needing to make an additional call to check the validity of the original pointer. In the case of a pointer error occurring, *dtksh* returns an empty string for the pointer value, and sets the **Tt_status** code accordingly. The **Tt_status** value is returned in the *status* argument. The **Tt_status** value is a string representing the error, and can assume any of the values shown in **<Tt/tt_c.h>**.

Some of the commands take a message scope as an argument. For these commands, the *scope* argument can be set to a string representing any of the constants documented for *tt_message_scope*, and in the manual pages for the individual ToolTalk functions.

`tt_file_netfile` *`variable status file name`*

`tt_netfile_file` *`variable status netfile name`*

`tt_host_file_netfile` *`variable status host file name`*

`tt_host_netfile_file` *`variable status host netfile name`*

`ttdt_open` *`variable status variable2 toolname vendor version sendStarted`*

This command returns in the *variable* argument the *procId* associated with the connection. It returns the file descriptor associated with the connection in *variable2*; this file descriptor can be used in registering an alternative Xt input handler via the *XtAddInput* command. The *sendStarted* argument is True or False. Any *procIds* returned by *ttdt_open* contain embedded spaces. To prevent *dtksh* from interpreting the *procId* as multiple arguments (versus a single argument with embedded spaces), references to the environment variable containing the *procId* must be within double quotes, as shown:

```
ttdt_close STATUS "$PROC_ID" "" True
```

`tttk_Xt_input_handler` *`procId source id`*

In order for the ToolTalk messages to be received and processed, the shell script must register an Xt input handler for the file descriptor returned by the call to *ttdt_open*. The Xt input handler is registered using the *XtAddInput* command, and the handler must be registered as a raw input handler. The input handler that the shell script registers should invoke *tttk_Xt_input_handler* to get the message received and processed. The following code block demonstrates how this is done:

```
ttdt_open PROC_ID STATUS FID "Tool" "HP" "1.0" True
XtAddInput INPUT_ID –r $FID "ProcessTTInput \"$PROC_ID\""
ProcessTTInput()
{
       tttk_Xt_input_handler $1 $INPUT_SOURCE $INPUT_ID
}
```

Refer to the description of the *XtAddInput* command for more details about alternative Xt input handlers. This command can be specified as an alternative Xt input handler, using the *XtAddInput* command. The *procId* value should be that which was returned by the *ttdt_open* command. When registering *tttk_Xt_input_handler* as an alternative Xt input handler, it must be registered as a raw handler to prevent *dtksh* from automatically breaking up the input into lines. This can be done as follows:

```
XtAddInput returnId –r $tt_fd \
       "tttk_Xt_input_handler \"$procId\""
```

The \" characters before and after the reference to the *procId* environment variable are necessary to protect the embedded spaces in the *procId* environment variable.

`ttdt_close` *`status procId newProcId sendStopped`*

This command closes the indicated communications connection, and optionally sends a *Stopped* notice, if the *sendStopped* argument is set to True. Because the *procId* returned by the call to *ttdt_open* contains embedded spaces, it must be enclosed within double quotes, as shown:

```
ttdt_close STATUS "$PROC_ID" "$NEW_PROC_ID" False
```

`ttdt_session_join` *`variable status sessId shellWidgetHandle join`*

This command joins the session indicated by the *sessId* argument. If the *sessId* argument does not specify a value (that is, it is an empty string), then the default session is joined. If the *shellWidgetHandle* argument specifies a widget handle (that is, it is not an empty string), then it should refer to a mappedWhenManaged applicationShellWidget. The *join* argument is True or False. This command returns an opaque pattern handle in the *variable* argument; this handle can be destroyed using the *ttdt_session_quit* command when it is no longer needed.

`ttdt_session_quit` *`status sessId sessPatterns quit`*

This command destroys the message patterns specified by the *sessPatterns* argument, and, if the *quit* argument is set to True, it quits the session indicated by the *sessId* argument, or it quits the default session if *sessId* is empty.

`ttdt_file_join` *`variable status pathName scope join dtksh-command`*

An opaque pattern handle is returned in the *variable* argument; this should be destroyed by calling *ttdt_file_quit* when there is no interest in monitoring messages for the indicated file. The requested *dtksh-command* is evaluated any time a message is received for the indicated file. When this *dtksh-command* is evaluated, the following environment variables are defined, and provide additional information about the received message:

DT_TT_MSG
: The opaque handle for the incoming message.

DT_TT_OP
The string representing the operation to be performed; that is, TTDT_DELETED, TTDT_MODIFIED, TTDT_REVERTED, TTDT_-MOVED or TTDT_SAVED.

DT_TT_PATHNAME
The pathname for the file to which this message pertains.

DT_TT_SAME_EUID_EGID
Set to True if the message was sent by an application operating with the same effective user ID and effective group ID as this process.

DT_TT_SAME_PROCID
Set to True if the message was sent by an application with the same *procId* (as returned by *ttdt_open*).

When the callback completes, it must indicate whether the passed-in message was consumed (replied-to, failed or rejected). If the callback returns the message (as passed in the *DT_TT_MSG* environment variable), it is assumed that the message was not consumed. If the message was consumed, the callback should return zero, or one of the values returned by the *tt_error_pointer* command. The callback can return its value in the following fashion:

```
return $DT_TT_MSG (or) return 0
```

`ttdt_file_quit` *`status patterns quit`*

This command destroys the message patterns specified by the *patterns* argument, and also unregisters interest in the pathname that was passed to the *ttdt_file_join* command if *quit* is set to True; the *patterns* argument should be the value returned by a call to the *ttdt_file_join* command.

`ttdt_file_event` *`status op patterns send`*

This command creates, and optionally sends, a ToolTalk notice announcing an event pertaining to a file. The file is indicated by the pathname passed to the *ttdt_file_join* command when *patterns* was created. The *op* argument indicates what should be announced for the indicated file, and can be set to TTDT_MODIFIED, TTDT_SAVED or TTDT_REVERTED. If *op* is set to TTDT_MODIFIED, this command registers to handle *Get_Modified*, *Save* and *Revert* messages in the scope specified when the *patterns* was created. If *op* is set to TTDT_SAVED or TTDT_REVERTED, this command unregisters from handling *Get_Modified*, *Save* and *Revert* messages for this file. If the *send* argument is set to True, the indicated message is sent.

`ttdt_Get_Modified` *`pathName scope timeout`*

This commands sends a *Get_Modified* request in the indicated scope, and waits for a reply, or for the specified *timeout* (in milliseconds) to elapse. It returns a value that can be used in a conditional. A value of True is returned if an affirmative reply is received within the specified *timeout*; otherwise, False is returned.

`ttdt_Save` *`status pathName scope timeout`*

This command sends a *Save* request in the indicated scope, and waits for a reply, or for the indicated *timeout* (in milliseconds) to elapse. A status of **TT_OK** is returned if an affirmative reply is received before the *timeout* elapses; otherwise, a **Tt_status** error value is returned.

`ttdt_Revert` *`status pathName scope timeout`*

This command sends a *Revert* request in the indicated scope, and waits for a reply, or for the indicated *timeout* (in milliseconds) to elapse. A status of **TT_OK** is returned if an affirmative reply is received before the *timeout* elapses; otherwise, a **Tt_status** error value is returned.

The following commands are typically used by the callback registered with the *ttdt_file_join* command. They serve as the mechanism for consuming and destroying a message. A message is consumed by either rejecting, failing or replying to it. The *tt_error_pointer* is used by the callback to get a return pointer for indicating an error condition.

`tt_error_pointer` *`variable ttStatus`*

This command returns a magic value, which is used by ToolTalk to represent an invalid pointer. The magic value returned depends on the *ttStatus* value passed in. Any of the valid **Tt_status** values can be specified.

`tttk_message_destroy` *`status msg`*

This command destroys any patterns that may have been stored on the message indicated by the *msg* argument, and then it destroys the message.

`tttk_message_reject` *`status msg msgStatus msgStatusString destroy`*

This command sets the status and the status string for the indicated request message, and then rejects the message. It then destroys the passed-in message if the *destroy* argument is set to True. This command is one way in which the callback specified with the *ttdt_file_join* command consumes a message. After rejecting the message, it is typically safe to destroy the message using *tttk_message_destroy*.

`tttk_message_fail` *`status msg msgStatus msgStatusString destroy`*

This command sets the status and the status string for the indicated request message, and then it fails the message. It destroys the passed-in message if the *destroy* argument is set to True. This command is one way in which the callback specified with the *ttdt_file_join* command consumes a message. After failing the message, it is typically safe to destroy the message, using *tttk_message_destroy*.

`tt_message_reply` *`status msg`*

This command informs the ToolTalk service that the shell script has handled the message specified by the *msg* argument. After replying to a message, it is typically safe to destroy the message using the *tttk_message_destroy* command.

Listing Widget Information

The *DtWidgetInfo* command provides the shell programmer a mechanism for obtaining information about the current set of instantiated widgets and their resources; the information is written to the standard output. This provides useful debugging information by including:

- The list of instantiated widgets, including: the name, class and parent of the widget; a handle for the widget; the name of the environment variable supplied when the widget was created; the state of the widget.
- The list of resources supported by a particular widget class.
- The list of constraint resources supported by a particular widget class.

DtWidgetInfo is called by using any of the following syntaxes; all of the arguments are optional:

`DtWidgetInfo [`*`widgetHandle`* | *`widgetName`*`]`

If no arguments are supplied, information about all existing widgets is written to standard output; the information includes the name, the handle, the environment variable, the parent, the widget class and the state. If arguments are supplied, they should be either widget handles, or the names of existing widgets; in this case, the information is written only for the requested set of widgets.

`DtWidgetInfo –r [`*`widgetHandle`* | *`widgetClass`*`]`

If no arguments are supplied, the list of supported resources is written to standard output for all available widget classes. If arguments are supplied, they should be either widget handles, or the widget class names; in this case, the information is written only for the requested set of widgets or widget classes.

`DtWidgetInfo –R [`*`widgetHandle`* | *`widgetClass`*`]`

If no arguments are supplied, the list of supported constraint resources, if any, is written to standard output for all available widget classes. If arguments are supplied, they should be either widget handles, or the widget class names; in this case, the information is written only for the requested set of widgets or widget classes.

`DtWidgetInfo –c [`*`widgetClass`*`]`

If no arguments are supplied, the list of supported widget class names is written to standard output. If arguments are supplied, *dtksh* writes the widget class name (if it is defined); otherwise, it writes an error message to standard error.

`DtWidgetInfo –h [`*`widgetHandle`*`]`

If no arguments are supplied, the list of active widget handles is written to standard output. If arguments are supplied, they should represent widget handles, in which case the name of the associated widget is written to standard output.

Convenience Functions

The XCDE system includes a file of *dtksh* convenience functions. This file is itself a shell script containing shell functions that may be useful to a shell programmer.

Before a shell script can access these functions, the shell script must first include the file containing the convenience functions. The convenience functions are located in the file **/usr/dt/lib/dtksh/DtFuncs.dtsh**, and are included in a shell script using the following notation:

```
. /usr/dt/lib/dtksh/DtFuncs.dtsh
```

DtkshAddButtons

This convenience function adds one or more buttons of the same kind into a composite widget. Most frequently, it is used to add a collection of buttons into a menupane or menubar.

`DtkshAddButtons` *`parent widgetClass label1 callback1`* `[`*`label2 callback2 ...`*`]`

`DtkshAddButtons [–w] `*`parent widgetClass variable1 label1 callback1`* `[`*`variable2 label2 callback2 ...`*`]`

The **–w** option indicates that the convenience function should return the widget handle for each of the buttons it creates. The widget handle is returned in the specified environment variable. The *widgetClass* argument can be set to any one of the following, and defaults to *XmPushButtonGadget*, if not specified:

XmPushButton
XmPushButtonGadget
XmToggleButton
XmToggleButtonGadget
XmCascadeButton
XmCascadeButtonGadget

Examples:

```
DtkshAddButtons $MENU XmPushButtonGadget Open do_Open Save \
       do_Save Quit exit

DtkshAddButtons –w $MENU XmPushButtonGadget B1 Open \
       do_Open B2 Save do_Save
```

DtkshSetReturnKeyControls

This convenience function configures a text widget (within a form widget), so the <carriage-return> key does not activate the default button within the form. Instead, the <carriage-return> key moves the focus to the next text widget within the form. This is useful if a window, containing a series of text widgets and the default button, should not be activated until the user presses the <carriage-return> key while the focus is in the last text widget.

`DtkshSetReturnKeyControls` *`textWidget nextTextWidget formWidget defaultButton`*

The *textWidget* argument specifies the widget to be configured so it catches the <carriage-return> key, and forces the focus to move to the next text widget (as indicated by the *nextTextWidget* argument). The *formWidget* argument specifies the form containing the default button, and must be the parent of the two text widgets. The *defaultButton* argument indicates which component to treat as the default button within the form widget.

Examples:

```
DtkshSetReturnKeyControls $TEXT1 $TEXT2 $FORM $OK

DtkshSetReturnKeyControls $TEXT2 $TEXT3 $FORM $OK
```

DtkshUnder, DtkshOver, DtkshRightOf, DtkshLeftOf

These convenience functions simplify the specification of certain classes of form constraints. They provide a convenient way of attaching a component to one edge of another component. They are used when constructing the resource list for a widget. This behaviour is accomplished using the ATTACH_WIDGET constraint.

`DtkshUnder` *`widgetId`* **[***`offset`***]**

`DtkshOver` *`widgetId`* **[***`offset`***]**

`DtkshRightOf` *`widgetId`* **[***`offset`***]**

`DtkshLeftOf` *`widgetId`* **[***`offset`***]**

The *widgetId* argument specifies the widget to which the current component is to be attached. The *offset* value is optional, and defaults to zero, if not specified.

Example:

```
XtCreateManagedWidget BUTTON4 button4 pushButton $FORM \
       labelString:"Exit" $(DtkshUnder $BUTTON2) \
       $(DtkshRightOf $BUTTON3)
```

DtkshFloatRight, DtkshFloatLeft, DtkshFloatTop, DtkshFloatBottom

These convenience functions simplify the specification of certain classes of form constraints. They provide a convenient way of positioning a component, independent of the other components within the form. As the form grows or shrinks, the component maintains its relative position within the form. The component may still grow or shrink, depending on the other form constraints specified for the component. This behaviour is accomplished using the ATTACH_POSITION constraint.

`DtkshFloatRight [`*`position`*`]`

`DtkshFloatLeft [`*`position`*`]`

`DtkshFloatTop [`*`position`*`]`

`DtkshFloatBottom [`*`position`*`]`

The optional *position* argument specifies the relative position to which the indicated edge of the component is positioned. A default position is used, if one is not specified.

Example:

```
XtCreateManagedWidgetBUTTON1 button1 pushButton \
       $FORM labelString:"Ok" $(DtkshUnder $SEPARATOR) \
       $(DtkshFloatLeft 10) $(DtkshFloatRight 40)
```

DtkshAnchorRight, DtkshAnchorLeft, DtkshAnchorTop, DtkshAnchorBottom

These convenience functions simplify the specification of certain classes of form constraints. They provide a convenient way of attaching a component to one of the edges of a form widget in such a fashion that, as the form grows or shrinks, the component's position does not change. However, depending on the other form constraints set on this component, the component may still grow or shrink in size. This behaviour is accomplished using the ATTACH_FORM constraint.

`DtkshAnchorRight [`*`offset`*`]`

`DtkshAnchorLeft [`*`offset`*`]`

`DtkshAnchorTop [`*`offset`*`]`

`DtkshAnchorBottom [`*`offset`*`]`

The optional *offset* argument specifies how far from the edge of the form widget the component should be positioned. If an offset is not specified, zero is used.

Example:

```
XtCreateManagedWidget BUTTON1 button1 pushButton \
      $FORM labelString:"Ok" $(DtkshUnder $SEPARATOR) \
      $(DtkshAnchorLeft 10) $(DtkshAnchorBottom 10)
```

DtkshSpanWidth, DtkshSpanHeight

These convenience functions simplify the specification of certain classes of form constraints. They provide a convenient way of configuring a component such that it spans either the full height or width of the form widget. This behaviour is accomplished by attaching two edges of the component (top and bottom for *DtkshSpanHeight*, and left and right for *DtkshSpanWidth*) to the form widget. The component typically resizes whenever the form widget is resized. The ATTACH_FORM constraint is used for all attachments.

`DtkshSpanWidth [`*`leftOffset rightOffset`*`]`

`DtkshSpanHeight [`*`topOffset bottomOffset`*`]`

The optional *offset* arguments specify how far from the edges of the form widget the component should be positioned. If an offset is not specified, zero is used.

Example:

```
XtCreateManagedWidget SEP sep separator $FORM $(DtkshSpanWidth 1 1)
```

DtkshDisplayInformationDialog, DtkshDisplayQuestionDialog, DtkshDisplayWarning-Dialog, DtkshDisplayWorkingDialog, DtkshDisplayErrorDialog

These convenience functions create a single instance of each of the Motif feedback dialogs. If an instance of the requested type of dialog already exists, it is reused. The parent of the dialog is obtained from the environment variable, *TOPLEVEL*, which should be set by the calling shell script, and then should not be changed. The handle for the requested dialog is returned in one of the following environment variables:

DTKSH_ERROR_DIALOG_HANDLE
DTKSH_QUESTION_DIALOG_HANDLE
DTKSH_WORKING_DIALOG_HANDLE
DTKSH_WARNING_DIALOG_HANDLE
DTKSH_INFORMATION_DIALOG_HANDLE

When attaching callbacks to the dialog buttons, the application should not destroy the dialog; it should simply unmanage the dialog so that it can be used again later. If it is necessary to destroy the dialog, the associated environment variable should also be cleared, so the convenience function does not attempt to reuse the dialog.

`DtkshDisplay*Dialog title message [`*`okCallback closeCallback \`*
`helpCallback dialogStyle``]`

The Ok button is always managed, and by default unmanages the dialog. The Cancel and Help buttons are only managed when a callback is supplied for them. The *dialogStyle* argument accepts any of the standard resource settings supported by the associated bulletin board resource.

Example:

```
DtkshDisplayErrorDialog "Read Error" "Unable to read the file" \
    "OkCallback" "CancelCallback" "" DIALOG_PRIMARY_APPLICATION_MODAL
```

DtkshDisplayQuickHelpDialog, DtkshDisplayHelpDialog

These convenience functions create a single instance of each of the help dialogs. If an instance of the requested type of help dialog already exists, it is reused. The parent of the dialog is obtained from the environment variable, *TOPLEVEL*, which should be set by the calling shell script, and then should not be changed. The handle for the requested dialog is returned in one of the following environment variables:

DTKSH_HELP_DIALOG_HANDLE
DTKSH_QUICK_HELP_DIALOG_HANDLE

If it is necessary to destroy a help dialog, the application should also clear the associated environment variable, so that the convenience function does not attempt to reuse the dialog.

`DtkshDisplay*HelpDialog title helpType helpInformation [`*`locationId`*`]`

The meaning of the arguments depends on the value specified for the *helpType* argument. The meanings are explained in the following table:

helpType	***helpInformation***	***locationId***
HELP_TYPE_DYNAMIC_STRING	help string	<not used>
HELP_TYPE_FILE	help file name	<not used>
HELP_TYPE_MAN_PAGE	manual page name	<not used>
HELP_TYPE_STRING	help string	<not used>
HELP_TYPE_TOPIC	help volume name	help topic location ID

Example:

```
DtkshDisplayHelpDialog "Help On Dtksh" HELP_TYPE_FILE "helpFileName"
```

Dtksh App-Defaults File

The *dtksh* app-defaults file, named **dtksh**, is in a location based on the following path description:

```
/usr/dt/app-defaults/<LANG>
```

The only information contained in this app-defaults file is the inclusion of the standard desktop base app-defaults file. The contents of the **dtksh** app-defaults file is as follows:

```
#include "Dt"
```

Non-String Values

The C bindings of the interfaces to X, Xt and Motif include many non-string values defined in headers. For example, the constraint values for a child of a form widget are declared, such as XmATTACH_FORM, with an Xt or Xm prefix followed by a descriptive name. Equivalent values are specified in *dtksh* by omitting the prefix, just as in an app-defaults file. For example: XmDIALOG_COMMAND_TEXT becomes DIALOG_COMMAND_TEXT; XtATTACH_FORM becomes ATTACH_FORM.

A Boolean value can be specified as an argument to a *dtksh* command using the words **True** or **False**; case is not significant.

Return Values From Built-in Commands

Graphical commands in *dtksh* fall into one of four categories, based on the definition of the corresponding C function in a windowing library:

1. The function returns no values. Example: *XtMapWidget*.
2. The function is void, but returns one or more values through reference arguments. Example: *XmGetColors*.
3. The function returns a non-Boolean value. Example: *XtCreateManagedWidget*.
4. The function returns a Boolean value. Example: *XtIsSensitive*.

A category 1 command follows the calling sequence of its corresponding C function exactly; the number and order of arguments can be determined by looking at the standard documentation for the function. Example:

```
XtMapWidget $FORM
```

A category 2 command also generally follows the calling sequence as its corresponding C function. Where a C caller would pass in a pointer to a variable in which a value is returned, the *dtksh* command returns a value in an environment variable. Example:

```
XmGetColors $FORM $BG FOREGROUND TOPSHADOW BOTTOMSHADOW SELECT
echo "Foreground color = " $FOREGROUND
```

A category 3 command differs slightly from its corresponding C function. Where the C function returns its value as the value of the procedure call, a *dtksh* command requires an additional argument, which is always the first argument, and is the name of an environment variable into which the return value is placed. Example:

```
XmTextGetString TEXT_VALUE $TEXT_WIDGET
echo "The value of the text field is "$TEXT_VALUE
```

A category 4 command returns a Boolean value that can be used in a conditional expression, just as with the corresponding C function. If the C function also returns values through reference variables (as in category 2), the *dtksh* command also uses variable names for the corresponding arguments. Example:

```
if XmIsTraversable $PUSH_BUTTON; then
       echo "The pushbutton is traversable"
else
       echo "The pushbutton is not traversable"
fi
```

Generally, the order and type of arguments passed to a command matches those passed to the corresponding C function, except as noted for category 3 commands. Other exceptions are described in the applicable command descriptions.

Widget Handles

Where a C function returns a widget handle, the corresponding *dtksh* commands set an environment variable equal to the widget handle. These are category 3 commands; they take as one of their arguments the name of an environment variable in which to return the widget handle.

For example, either of the following commands could be used to create a new form widget; in both cases, the widget handle for the new form widget is returned in the environment variable *FORM*:

```
XtCreateManagedWidget FORM name XmForm $PARENT

XmCreateForm FORM $PARENT name
```

After either of the above commands, *$FORM* can be used to reference the form widget. For instance, to create a label widget within the form widget just created, the following command could be used:

```
XmCreateLabel LABEL $FORM namelabelString:"Hi Mom" \
       topAttachment:ATTACH_FORM leftAttachment:ATTACH_FORM
```

There is a special widget handle called NULL, provided for cases where a shell script may need to specify a NULL widget. For example, the following disables the **defaultButton** resource for a form widget:

```
XtSetValues $FORM defaultButton:NULL
```

Widget Resources

Some of the Xt and Motif commands allow the shell script to pass in a variable number of arguments, representing resource and value pairs. This is similar to the *arglist* passed in to the corresponding Xt or Motif C function. Examples of these commands include any of the commands used to create a widget, and the *XtSetValues* command. In *dtksh*, resources are specified by a string with the following syntax: *resource*:*value*.

The name of the resource is given in the resource portion of the string; it is constructed by taking the corresponding Xt or Motif resource name and omitting the Xt or Xm prefix. The value to be assigned to the resource is given in the value portion of the string. The *dtksh* utility automatically converts the value string to an appropriate internal representation. For example:

```
XtSetValues $WIDGET height:100 width:200 resizePolicy:RESIZE_ANY
XmCreateLabel LABEL $PARENT myLabel labelString:"Close Dialog"
```

When widget resources are retrieved using *XtGetValues*, the return value has the same syntax. For example:

```
XtGetValues $WIDGET height:HEIGHT resizePolicy:POLICY \
       sensitive:SENSITIVE
echo $HEIGHT
echo $POLICY
echo $SENSITIVE
```

Certain types of resource values have special representation. These include string tables and bit masks. For instance, the XmList widget allows a string table to be specified both for the items and the **selectedItems** resources. In *dtksh*, a string table is represented as a comma-separated list of strings, which is compatible with how Motif handles them from a resource file. When a resource that returns a string table is queried using *XtGetValues*( ), the resulting value is again a comma-separated set of strings. A resource that expects a bit mask value to be passed in, expects the mask to be specified as a string composed of the various mask values separated by the ‘‘|’’ character. When a resource that returns a bit mask is queried, the return value also is a string representing the enabled bits, separated by the ‘‘|’’ character. For example, the following sets the **mwmFunctions** resource for the *VendorShell* widget class:

```
XtSetValues mwmFunctions MWM_FUNC_ALL|MWM_FUNC_RESIZE
```

Unsupported Resources

The *dtksh* utility supports most of the resources provided by Motif; however, there are certain resources that *dtksh* does not support. The list of unsupported resources follows. Several of these resources can be specified at widget creation time by using *XtSetValues*, but cannot be retrieved using *XtGetValues*; these are indicated by the asterisk (*) following the resource name.

All Widget And Gadget Classes:
- Any font list resource (*)
- Any pixmap resource (*)

Composite:
- *insertPosition*
- children

Core:
- accelerators
- translations (*)
- colourmap

XmText:
- *selectionArray*
- *selectionArrayCount*

ApplicationShell:
- *argv*

WMShell:
- *iconWindow*
- *windowGroup*

Shell:
- *createPopupChildrenProc*

XmSelectionBox:
- *textAccelerators*

Manager, Primitive and *Gadget Subclasses:*
- *userData*

XmFileSelectionBox:
- *dirSearchProc*
- *fileSearchProc*
- *qualifySearchDataProc*

EXIT STATUS

See *sh* in the X/Open CAE Specification, **Commands and Utilities**, **Issue 4, Version 2**.

CONSEQUENCES OF ERRORS

See *sh* in the X/Open CAE Specification, **Commands and Utilities**, **Issue 4, Version 2**.

APPLICATION USAGE

Initialising The Toolkit Environment

Before any of the Xlib, Xt or Motif commands can be invoked, the shell script must first initialise the Xt toolkit by invoking the *XtInitialize* command, which returns an application shell widget. *XtInitialize*, as with all of the commands that return a widget handle, returns the widget handle in the environment variable named in its first argument. For example:

```
XtInitialize TOPLEVEL myShellName Dtksh $0$@
```

Shell script writers should invoke the *XtInitialize* command as one of the first commands within a *dtksh* shell script. This allows *dtksh* to locate its message catalogue and the correct app-defaults file. If a shell error occurs before *XtInitialize* has been called, it is possible that unlocalised error messages may be displayed. The *dtksh* utility provides a default app-defaults file to use if the call to *XtInitialize* specifies an application class name of *Dtksh*. This app-defaults file loads in the standard set of desktop application default values, so that these applications have a consistent look with other desktop applications.

Cooperating with WorkSpace Management

The *dtksh* utility provides access to all of the major workspace management functions of the desktop libraries, including functions for:

- Querying and setting the set of workspaces with which an application is associated.
- Querying the list of all workspaces.
- Querying and setting the current workspace.
- Requesting that an application be notified any time the user changes to a different workspace.

From a user's perspective, workspaces are identified by a set of names, but from the workspace manager's perspective, workspaces are identified by X atoms. Whenever the shell script asks for a list of workspace identifiers, a string of X atoms is returned; if more than one X atom is present, the list is comma-separated.

The workspace manager expects that the shell script uses the same format when passing workspace identifiers back to it. During a given session, it is safe for the shell script to work with the X atoms since they remain constant over the lifetime of the session. However, as was shown in the Session Management shell script example, if the shell script is going to save and restore workspace identifiers, the workspace identifiers must be converted from their X atom representation to a string before they are saved. Then, when the session is restored, the shell script needs to remap the names into X atoms before passing the information on to the workspace manager. Mapping between X atoms and strings and between strings and X atoms uses the following two commands:

```
XmInternAtom ATOM $DISPLAY $WORKSPACE_NAME false
XmGetAtomName NAME $DISPLAY $ATOM
```

Creating Localised Shell Scripts

Scripts written for *dtksh* are internationalised, and then localised, in a process very similar to C applications. All strings that may be presented to the user are identified in the script; a post-processor extracts the strings from the script, and from them builds a catalogue, which can then be translated to any desired locale. When the script executes, the current locale determines which message catalogue is searched for strings to display. When a string is to be presented, it is identified by a message-set ID (corresponding to the catalogue), and a message number within the set; these values determine what text the user sees. The following code illustrates the process:

```
# Attempt to open our message catalog
catopen MSG_CAT_ID "myCatalog.cat"

# The localized button label is in set 1, and is message # 2
XtCreatePushButton OK $PARENT ok
labelString:$(catgets $MSG_CAT_ID 1 2 "OK")

# The localized button label is in set 1, and is message #3
XtCreatePushButton CANCEL $PARENT cancel
labelString:$(catgets $MSG_CAT_ID 1 3 "Cancel")

# Close the message catalog, when no longer needed
catclose $MSG_CAT_ID
```

The file descriptor returned by *catopen* must be closed using *catclose*, and not using the *sh exec* command.

Using the dtksh Utility to Access X Drawing Functions

The commands of the *dtksh* utility include standard Xlib drawing functions to draw lines, points, segments, rectangles, arcs and polygons. In the standard C programming environment, these functions take a graphics context, or GC as an argument, in addition to the drawing data. In *dtksh* drawing functions, a collection of GC options are specified in the argument list to the command. By default, the drawing commands create a GC that is used for that specific command and then discarded. If the script specifies the **–gc** option, the name of the graphics context object can be passed to the command; this GC is used in interpreting the command, and the variable is updated with any modifications to the GC performed by the command.

–**gc** *GC*
: *GC* is the name of an environment variable that has not yet been initialised, or which has been left holding a graphic context by a previous drawing command. If this option is specified, it must be the first *GC* option specified.

–**foreground** *color*
: Foreground colour, which can be either the name of a colour or a pixel number.

–**background** *color*
: Background colour, which can be either the name of a colour or a pixel number.

–**font** *font name*
: Name of the font to be used.

–**line_width** *number*
: Line width to be used during drawing.

–**function** *drawing function*
: Drawing function, which can be any of the following: **xor**, **or**, **clear**, **and**, **copy**, **noop**, **nor**, **nand**, **set**, **invert**, **equiv**, **andReverse**, **orReverse** or **copyInverted**.

–**line_style** *style*
Line style, which can be any of the following: **LineSolid**, **LineDoubleDash** or **LineOnOffDash**.

Setting Widget Translations:

The *dtksh* utility provides mechanisms for augmenting, overriding and removing widget translations, much as in the C programming environment. In C, an application installs a set of translation action procedures, which can then be attached to specific sequences of events (translations are composed of an event sequence and the associated action procedure). Translations within *dtksh* are handled in a similar fashion, except only a single action procedure is available. This action procedure, named *ksh_eval*, interprets any arguments passed to it as *dtksh* commands, and evaluates them when the translation is triggered. The following shell script segment gives an example of how translations can be used:

```
BtnDownProcedure()
{
  echo "Button Down event occurred in button "$1
}
XtCreateManagedWidget BUTTON1 button1 XmPushButton $PARENT
  labelString:"Button 1"
  translations:'#augment
      <EnterNotify>:ksh_eval("echo Button1 entered")
      <Btn1Down>:ksh_eval("BtnDownProcedure 1")'
XtCreateManagedWidget BUTTON2 button2 XmPushButton $PARENT
  labelString:"Button 2"
XtOverrideTranslations $BUTTON2
         '#override
      <Btn1Down>:ksh_eval("BtnDownProcedure 2")'
```

EXAMPLES
None.

SEE ALSO
sh in the X/Open CAE Specification, **Commands and Utilities**, **Issue 4, Version 2**.

CHANGE HISTORY
First released in Issue 1.

Chapter 12

Terminal Emulation Services

12.1 Introduction

The XCDE terminal emulation services provide a window for applications written for character-based terminals such as the VT220. These applications include XPG4 commands, shell scripts, and other applications not written for an X Windows environment. The services are available in two forms: a stand-alone client and a widget that can be used inside applications that require terminal emulation.

The XCDE terminal emulation services are consistent with the referenced ANSI X3.64-1979 standard and the referenced ISO/IEC 6429:1992 standard. The services are based in part on the X Consortium's *xterm* and popular terminals compatible with the VT220.

The user interface is consistent with the **OSF/Motif Style Guide**, so the client and widget blend well on a screen populated by windowed applications.

12.2 Functions

This section defines the functions, macros and external variables that provide XCDE terminal emulation services to support application portability at the C-language source level.

NAME

DtCreateTerm — create a DtTerm widget

SYNOPSIS

```
#include <Dt/Term.h>

Widget DtCreateTerm(Widget parent,
                    String name,
                    ArgList arglist,
                    Cardinal argcount);
```

DESCRIPTION

The *DtCreateTerm*( ) function creates a terminal emulator widget hierarchy.

The *parent* argument specifies the parent widget ID.

The *name* argument specifies the name of the created widget.

The *arglist* argument specifies the argument list.

The *argcount* argument specifies the number of attribute and value pairs in the argument list (*arglist*).

RETURN VALUE

Upon successful completion, the *DtCreateTerm*( ) function returns the DtTerm widget ID.

SEE ALSO

dtterm, Section 12.7 on page 283; **<Dt/Term.h>**, *DtTerm*, *DtTermInitialize*( ), *DtTermDisplaySend*( ), *DtTermSubprocSend*( ), *DtTermSubprocReap*( ).

CHANGE HISTORY

First released in Issue 1.

NAME

DtTermDisplaySend — send data to a DtTerm widget's display

SYNOPSIS

```
#include <Dt/Term.h>

void DtTermDisplaySend(Widget widget,
                       unsigned char *buffer,
                       int length);
```

DESCRIPTION

The *DtTermDisplaySend*( ) function sends data to a DtTerm widget's display.

The *widget* argument specifies the DtTerm widget ID.

The *buffer* argument specifies the string (single- or multi-byte depending on the locale) to be sent to the display. The string may contain NULL bytes.

The *length* argument specifies the length of *buffer* in bytes.

The *DtTermDisplaySend*( ) function allows the program to write text directly to the DtTerm widget's text display. The text is handled in the same manner as text received from the child process. Before the text is written in the window, the DtTerm widget's input parser processes it.

RETURN VALUES

The *DtTermDisplaySend*( ) function returns no value.

SEE ALSO

<Dt/Term.h>, *DtTerm*.

CHANGE HISTORY

First released in Issue 1.

NAME

DtTermInitialise — prevent accelerators from being installed on a DtTerm widget

SYNOPSIS

```
#include <Dt/Term.h>

void DtTermInitialize();
```

DESCRIPTION

The *DtTermInitialize*( ) function prevents the XmBulletinBoard widget from installing accelerators on DtTerm widgets. It enables DtTerm widgets to receive certain key events, such as Return and Escape, normally not passed by Motif to XmPrimitive widgets.

RETURN VALUES

The *DtTermInitialize*( ) function returns no value.

APPLICATION USAGE

The application must call *DtTermInitialize*( ) before initialising the Xt Toolkit with *XtAppInitialize*( ).

SEE ALSO

<Dt/Term.h>, *DtTerm*, *XmPrimitive*, *XmBulletinBoard* in the X/Open CAE Specification, **Motif Toolkit API**; *XtAppInitialize*( ) in the X/Open CAE Specification, **Window Management: X Toolkit Intrinsics**.

CHANGE HISTORY

First released in Issue 1.

NAME

DtTermSubprocReap — allow a DtTerm widget to clean up after subprocess termination

SYNOPSIS

```
#include <Dt/Term.h>

void DtTermSubprocReap(pid_t pid,
                       int *stat_loc);
```

DESCRIPTION

The *DtTermSubprocReap*( ) function allows DtTerm widgets to function correctly in applications that have installed a SIGCHLD signal handler.

The *pid* argument specifies the process ID of the child process *wait*( ) returns.

The *stat_loc* argument specifies the termination information *wait*( ) returns.

RETURN VALUES

The *DtTermSubprocReap*( ) function returns no value.

APPLICATION USAGE

The *DtTermSubprocReap*( ) function allows an application to install its own SIGCHLD signal handler.

The application must install its own SIGCHILD signal handler and call *DtTermSubprocReap*( ) if the DtTerm widget was created with the **DtSubprocessTerminatorCatch** resource set to False.

The application must call the *DtTermSubprocReap*( ) function after performing a *wait*( ) (or associated function) on a terminated child process. If the child process is associated with a DtTerm widget, the widget's data structures are cleaned up and the associated callbacks invoked.

SEE ALSO

<Dt/Term.h>, *DtTerm*; *sigaction*( ), *wait*( ) in the X/Open CAE Specification, **System Interfaces and Headers**, **Issue 4, Version 2**.

CHANGE HISTORY

First released in Issue 1.

NAME

DtTermSubprocSend — send data to a DtTerm widget's subprocess

SYNOPSIS

```
#include <Dt/Term.h>

void DtTermSubprocSend(Widget widget,
                       unsigned char *buffer,
                       int length);
```

DESCRIPTION

The *DtTermSubprocSend*( ) function sends data to a DtTerm widget's child process.

The *widget* argument specifies the DtTerm widget ID.

The *buffer* argument specifies the string (single- or multi-byte depending on the locale) to be sent to the display. The string may contain NULL bytes.

The *length* argument specifies the length of *buffer* in bytes.

The *DtTermSubprocSend*( ) function allows the program to send commands to the DtTerm widget's child process. The characters are handled in the same manner as keyboard input.

RETURN VALUES

The *DtTermSubprocSend*( ) function returns no value.

SEE ALSO

<Dt/Term.h>, *DtTerm*.

CHANGE HISTORY

First released in Issue 1.

12.3 Widgets

This section defines the widget class that provides the widget used by the XCDE terminal emulation services to support application portability at the C-language source level.

NAME

DtTerm — DtTerm widget class

SYNOPSIS

```
#include <Dt/Term.h>
```

DESCRIPTION

The DtTerm widget provides the core set of functionality needed to emulate a terminal that supports the control sequences in the referenced ANSI X3.64-1979 standard and the referenced ISO/IEC 6429:1992 standard. This functionality includes text rendering, scrolling, margin and tab support, escape sequence parsing and the low-level implementation-specific interfaces required to allocate and configure a pseudo-terminal device and to update the system's implementation-dependent database of logged-in users (see *who* in the X/Open CAE Specification, **Commands and Utilities**, **Issue 4, Version 2**).

Classes

The DtTerm widget inherits behaviour and resources from the *Core* and *XmPrimitive* classes.

The class pointer is **dtTermWidgetClass**.

The class name is *DtTerm*.

New Resources

The following table defines a set of widget resources used by the application to specify data. The application can also set the resource values for the inherited classes to set attributes for this widget. To reference a resource by name or by class in a **.Xdefaults** file, the application must remove the **DtN** or **DtC** prefix and use the remaining letters. To specify one of the defined values for a resource in a **.Xdefaults** file, the application must remove the **Dt** prefix and use the remaining letters (in either lower case or upper case, but including any underscores between words). The codes in the access column indicate if the given resource can be set at creation time (C), set by using *XtSetValues*( ) (S), retrieved by using *XtGetValues*( ) (G), or is not applicable (N/A).

DtTerm Resource Set, Part 1 of 2

Name	Class	Type	Default	Access
DtNallowSendEvents	**DtCAllowSendEvents**	**Boolean**	False	CSG
DtNappCursorDefault	**DtCAppCursorDefault**	**Boolean**	False	CSG
DtNappKeypadDefault	**DtCAppKeypadDefault**	**Boolean**	False	CSG
DtNautoWrap	**DtCAutoWrap**	**Boolean**	True	CSG
DtNbackgroundIsSelect	**DtCBackgroundIsSelect**	**Boolean**	False	CG
DtNbaseHeight	**DtCBaseHeight**	**int**	0	G
DtNbaseWidth	**DtCBaseWidth**	**int**	0	G
DtNblinkRate	**DtCBlinkRate**	**int**	250	CSG
DtNc132	**DtCC132**	**Boolean**	False	CSG
DtNcharCursorStyle	**DtCCharCursorStyle**	**unsigned char**	DtTERM_CHAR-_CURSOR_BOX	CSG

DtTerm Resource Set, Part 2 of 2				
Name	**Class**	**Type**	**Default**	**Access**
DtNcolumns	**DtCColumns**	**short**	80	CSG
DtNconsoleMode	**DtCConsoleMode**	**Boolean**	False	CG
DtNemulationID	**DtCEmulationID**	**String**	''DtTermWidget''	CSG
DtNheightInc	**DtCHeightInc**	**int**	0	G
DtNinputVerifyCallback	**DtCCallback**	**XtCallbackList**	NULL	C
DtNjumpScroll	**DtCJumpScroll**	**Boolean**	True	CSG
DtNkshMode	**DtCKshMode**	**Boolean**	False	CSG
DtNlogging	**DtCLogging**	**Boolean**	False	CSG
DtNmapOnOutput	**DtCMapOnOutput**	**Boolean**	False	CSG
DtNmapOnOutputDelay	**DtCMapOnOutputDelay**	**int**	0	CSG
DtNmarginBell	**DtCMarginBell**	**Boolean**	False	CSG
DtNmarginHeight	**DtCMarginHeight**	**Dimension**	2	CSG
DtNmarginWidth	**DtCMarginWidth**	**Dimension**	2	CSG
DtNnMarginBell	**DtCNMarginBell**	**int**	10	CSG
DtNoutputLogCallback	**DtCCallback**	**XtCallbackList**	NULL	C
DtNpointerBlank	**DtCPointerBlank**	**Boolean**	False	CSG
DtNpointerBlankDelay	**DtCPointerBlankDelay**	**int**	2	CSG
DtNpointerColor	**DtCForeground**	**String**	dynamic	CSG
DtNpointerColorBackground	**DtCBackground**	**String**	dynamic	CSG
DtNpointerShape	**DtCPointerShape**	**String**	**xterm**	CSG
DtNreverseWrap	**DtCReverseWrap**	**Boolean**	False	CSG
DtNrows	**DtCRows**	**short**	24	CSG
DtNsaveLines	**DtCSaveLines**	**string**	4s	CG
DtNshadowType	**DtCShadowType**	**unsigned char**	DtSHADOW_IN	CSG
DtNstatusChangeCallback	**DtCCallback**	**XtCallbackList**	NULL	C
DtNsubprocessArgv	**DtCSubprocessArgv**	**String** *	NULL	CG
DtNsubprocessCmd	**DtCSubprocessCmd**	**String**	NULL	CG
DtNsubprocessExec	**DtCSubprocessExec**	**Boolean**	True	CSG
DtNsubprocessLoginShell	**DtCSubprocessLoginShell**	**Boolean**	False	CG
DtNsubprocessPid	**DtCSubprocessPid**	**int**	−1	G
DtNsubprocess-TerminationCallback	**DtCCallback**	**XtCallbackList**	NULL	C
DtNsubprocess-TerminationCatch	**DtCSubprocess-TerminationCatch**	**Boolean**	True	CSG
DtNsunFunctionKeys	**DtCSunFunctionKeys**	**Boolean**	False	CSG
DtNtermDevice	**DtCTermDevice**	**int**	−1	CG
DtNtermDeviceAllocate	**DtCTermDeviceAllocate**	**Boolean**	True	CG
DtNtermId	**DtCTermId**	**String**	''vt220''	CSG
DtNtermName	**DtCTermName**	**String**	''dtterm''	CSG
DtNtermSlaveName	**DtCTermSlaveName**	**String**	NULL	CG
DtNttyModes	**DtCTtyModes**	**String**	NULL	CSG
DtNuserBoldFont	**DtCUserBoldFont**	**XmFontList**	dynamic	CSG
DtNuserFont	**DtCUserFont**	**XmFontList**	dynamic	CSG
DtNverticalScrollBar	**DtCVerticalScrollBar**	**Widget**	NULL	CSG
DtNvisualBell	**DtCVisualBell**	**Boolean**	False	CSG
DtNwidthInc	**DtCWidthInc**	**int**	0	G

allowSendEvents

Specifies that the terminal emulator allow synthetic events (generated and sent by another application). Enabling this resource opens up a possible security hole.

appCursorDefault

Specifies the initial cursor mode. If True, the cursor keys are initially in application mode. If False, the cursor keys are initially in cursor mode.

appKeypadDefault
Specifies the initial keypad mode. If True, the keypad keys are initially in application mode. If False, the keypad keys are initially in numeric mode.

c132
Specifies whether or not the DECCOLM escape sequence that switches between a 132- and 80-column window is honoured.

consoleMode
Specifies that output directed at **/dev/console** be directed instead to the terminal window. It is provided as a way to prevent output that would normally be displayed on the internal terminal emulator (ITE) from overwriting the X server's display. It is not provided as a general mechanism to direct the output from an arbitrary system's **/dev/console** to an arbitrary X server. Ownership of, and read-write permission for, **/dev/console** is required to redirect console output.

DtNautoWrap
Specifies whether or not auto-wrap is initially enabled.

DtNbackgroundIsSelect
Controls the background colour. When False, the background colour is the colour specified. When True, the background colour is the select colour corresponding to the background and is constant with other Motif-based applications.

DtNbaseHeight
Specifies the terminal window's base height. With this resource the application computes its base height for the shell widget, which then allows the window manager to provide appropriate sizing feedback to the user. The height of the terminal window is:

DtNbaseHeight + DtNrows × DtNheightInc

DtNbaseWidth
Specifies the terminal window's base width. With this resource the application computes its base width for the shell widget, which then allows the window manager to provide appropriate sizing feedback to the user. The width of the terminal window is:

DtNbaseWidth + DtNcolumns × DtNwidthInc

DtNblinkRate
Specifies the number of milliseconds the cursor is in the on and off states while blinking. A value of 250 blinks the cursor two times per second. A value of zero turns blinking off.

DtNc132
This resource specifies whether or not the DECCOLM escape sequence should be honoured.

DtNcharCursorStyle
Specifies the text cursor shape. A DtTERM_CHAR_CURSOR_BOX value specifies a cursor the width and height of the base font's bounding box. A DtTERM_CHAR_CURSOR_BAR value specifies a cursor the width of the base font's bounding box, two pixels high, and drawn with its top on the baseline. The default is DtTERM_CHAR_CURSOR_BOX.

DtNcolumns
Specifies the number of text columns in the terminal window. For additional information, see **DtNbaseWidth**.

DtNemulationID
Specifies the string to which the *TERMINAL_EMULATOR* environment variable is set.

DtNheightInc
Specifies the character cell height used as the height increment when calculating the size of the terminal window. For additional information, see **DtNbaseHeight**.

DtNinputVerifyCallback
Specifies the list of callbacks called before the DtTerm widget sends text to the child process. The text may be generated either in response to keyboard input, selection or drag and drop. The **DtTermInputVerifyCallbackStruct** structure's address is passed to this callback. DtCR_TERM_INPUT_VERIFY is the *reason* set by the callback.

DtNjumpScroll
Specifies that the DtTerm widget uses jump scrolling. The maximum number of lines that may be jump scrolled is limited to the number of lines in the terminal window. The DtTerm widget guarantees that all lines are displayed.

DtNkshMode
Enables *ksh* mode. With *ksh* mode, a key pressed with the extend modifier bit set generates an escape character followed by the character generated by the unextended keystroke. This option is provided for use with *emacs* and the emacs command-line editor mode of *ksh*. It conflicts with the normal meta key use for generating extended single byte characters and for generating multi-byte Asian characters.

DtNmapOnOutput
Indicates that the terminal widget map (de-iconify) itself upon subprocess output if it is unmapped (iconified). An initial period of time during which the terminal widget does not map itself upon subprocess output can be specified via the **DtNmapOnOutputDelay** resource.

DtNmapOnOutputDelay
Specifies the number of seconds after start-up that the widget does not honour the **DtNmapOnOutput** resource. This allows for initial output (for example, shell prompts) to be sent to the terminal without auto-mapping the window. The default is zero (no delay).

DtNmarginBell
Specifies whether or not the bell rings when the user types near the right margin. The distance from the right margin is specified by the **nMarginBell** resource.

DtNmarginHeight
Specifies the height of the margin between the text and the top and bottom of the DtTerm widget's window.

DtNmarginWidth
Specifies the width of the margin between the text and both sides of the DtTerm widget's window.

DtNnMarginBell
Specifies the number of characters from the right margin at which the margin bell rings when enabled via the **DtNmarginBell** resource.

DtNoutputLogCallback
Specifies the list of callbacks called when the widget receives text from the child process. **DtTermOutputLogCallbackStruct** is the structure type whose address is passed to this callback. DtCR_TERM_OUTPUT_LOG is the *reason* set by the callback.

DtNpointerBlank
Indicates that the pointer cursor is in blanking mode. In this mode, the cursor turns on when the pointer is moved, and is blanked either after a selectable number of seconds or after keyboard input. The delay is set via the **DtNpointerBlankDelay** resource.

DtNpointerBlankDelay
Specifies the number of seconds to wait after the pointer has stopped moving before blanking the pointer (see **DtNpointerBlank**). A value of zero prevents the pointer from blanking until a key is pressed.

DtNpointerColor
Specifies the foreground colour the DtTerm widget uses for the terminal window's pointer (X11 cursor). The default is the terminal window's foreground colour.

DtNpointerColorBackground
Specifies the background colour the DtTerm widget uses for the terminal window's pointer (X11 cursor). The default is the terminal window's background colour.

DtNpointerShape
Specifies the X cursor font character the DtTerm widget uses as the pointer cursor. The font character should be specified as a string from the include file, **<X11/cursorfont.h>**, with the leading **XC_** removed.

DtNreverseWrap
Specifies whether or not reverse-wraparound is enabled.

DtNrows
Specifies the number of rows of text in the terminal window. For additional information, see **DtNbaseHeight**.

DtNsaveLines
Defines the number of lines in the terminal buffer beyond the length of the window. The resource value consists of a number followed by an optional suffix. If no suffix is included or the suffix is ''l'' (ell), the total length of the terminal buffer is the number plus the length of the terminal window. If the suffix is ''s,'' the total length of the terminal buffer is (the number plus one) times the length of the terminal window. The DtTerm widget tries to maintain the same buffer-to-window ratio when the window is resized larger.

DtNshadowType
Specifies the type of shadow drawn around the terminal window. See the XmFrame widget for supported shadow types.

DtNstatusChangeCallback
Specifies the list of callbacks called when the DtTerm widget's status changes. Status changes include changes such as cursor position, caps lock state and insert char state. The **DtTermStatusChangeCallbackStruct** structure's address is passed to this callback. DtCR_TERM_STATUS_CHANGE is the *reason* the callback sends.

DtNsubprocessArgv
Specifies the argument list passed to the subprocess if **DtNsubprocessExec** is True. If **DtNsubprocessCmd** is NULL, the first string of this argument is used as the name of the command to execute.

DtNsubprocessCmd
Specifies the name of the command to run if **DtNsubprocessExec** is True. If **DtNsubprocessExec** is NULL, the first string of the **DtNsubprocessArgv** argument is used.

DtNsubprocessExec
This resource specifies whether or not a subprocess is *fork*( ) and *exec*( ). If True, a subprocess is launched as specified via the **DtNsubprocessArgv** resource, if set, or the **DtNsubprocessCmd** resource, if set, or the *SHELL* environment variable, if set, or the default system shell.

DtNsubprocessloginShell Indicates that the starting shell is to be a login shell (that is, the first character of *argv*[0] is to be a dash), that tells the shell to read the system's **profile** and the user's **.profile** files (for *ksh* and *sh*).

DtNsubprocessPid
Supplies the process ID of the subprocess running in the terminal widget when **DtNsubprocessExec** is True.

DtNsubprocessTerminationCallback
Supplies the list of callbacks called when the subprocess associated with the DtTerm widget exits. The **DtTermSubprocessTerminationCallbackStruct** structure's address is passed to this callback. DtCR_TERM_SUBPROCESS_TERMINATION is the *reason* the callback sends.

DtNsubprocessTerminationCatch
Specifies whether or not the DtTerm widget installs a signal handler to catch the subprocess termination. If the application installs its own signal handler, the application must catch the subprocess termination and inform the DtTerm widget via the *DtTermSubprocReap*( ) function.

DtNsunFunctionKeys
Specifies whether or not Sun Function Key escape sequences are generated for function keys instead of standard escape sequences. See Section 12.7.3 on page 296 for a description of the Sun Function Key escape sequences.

DtNtermDevice
Supplies the file descriptor for the master side of the pseudo-terminal device associated with the DtTerm widget.

DtNtermDeviceAllocate
Specifies whether or not the DtTerm widget allocates a pseudo-terminal device, or uses the pseudo-terminal device passed to it via the **DtNtermDevice** resource.

DtNtermId
Supplies the name the DtTerm widget uses to select the correct response to terminal ID queries. Valid values are **vt100**, **vt101**, **vt102** and **vt220**.

DtNtermName
Supplies the name the DtTerm widget uses in setting the *TERM* environment variable. The default is *dtterm*.

DtNtermSlaveName
Supplies the name of the slave device of the pseudo-terminal device associated with the DtTerm widget. The DtTerm widget uses this resource to update the system's implementation-dependent database of logged-in users (see *who* in the X/Open CAE Specification, **Commands and Utilities**, **Issue 4, Version 2**) entry associated with the subprocess.

DtNttyModes
Specifies a string containing terminal-setting keywords and the characters to which they may be bound. Allowable keywords include: **intr**, **quit**, **erase**, **kill**, **eof**, **eol**, **swtch**, **start**, **stop**, **brk**, **susp**, **dsusp**, **rprnt**, **flush**, **weras** and **lnext**. The terminal emulator correctly parses and silently ignores keywords that do not apply to a specific architecture. Control characters can be specified as **^char** (for example, **^c** or **^u**), and **^?** can be used to indicate delete. This is useful for overriding the default terminal settings without having to do an *stty* every time a DtTerm widget is created.

DtNuserBoldFont
Supplies the **XmFontList** the DtTerm widget uses to display bold terminal text. The terminal emulator supports only character or mono-spaced fonts. When using proportional fonts, the behaviour is undefined. The terminal emulator generates a default bold font based on the XLFD name of the **userFont**. If that font is not available, the terminal emulator generates bold text by overstriking (with a one pixel offset) the **userFont**.

DtNuserFont
Supplies the **XmFontList** the DtTerm widget uses to display terminal text. The terminal emulator supports only character or mono-spaced fonts. When using proportional fonts, the behaviour is undefined. The terminal emulator gets a default font via the **XmNtextFontList** value of the parent bulletin board (see the XmBulletinBoard widget) in the same manner as the XmText widget.

DtNverticalScrollBar
Specifies a an application-supplied vertical scroll bar widget to update as scrolling occurs. The DtTerm widget does not create the scroll bar.

DtNvisualBell
Specifies whether the DtTerm widget uses a visible bell (that is, flashing) instead of an audible bell when <control>-G is received.

DtNwidthInc
Specifies the character cell width the DtTerm widget uses as the width increment when calculating the size of the terminal window. For additional information, see **DtNbaseWidth**.

Inherited Resources

The DtTerm widget inherits behaviour and resources from the following named superclasses. For a complete description of each resource, see the entry in X/Open CAE Specification, **Motif Toolkit API** for that superclass.

XmPrimitive Resource Set				
Name	**Class**	**Type**	**Default**	**Access**
XmNbottomShadowColor	**XmCBottomShadowColor**	**Pixel**	dynamic	CSG
XmNbottom-ShadowPixmap	**XmCBottom-ShadowPixmap**	**Pixmap**	XmUNSPECIFIED-_PIXMAP	CSG
XmNforeground	**XmCForeground**	**Pixel**	dynamic	CSG
XmNhelpCallback	**XmCCallback**	**XtCallbackList**	NULL	C
XmNhighlightColor	**XmCHighlightColor**	**Pixel**	dynamic	CSG
XmNhighlightOnEnter	**XmCHighlightOnEnter**	**Boolean**	False	CSG
XmNhighlightPixmap	**XmCHighlightPixmap**	**Pixmap**	dynamic	CSG
XmNhighlightThickness	**XmCHighlightThickness**	**Dimension**	2	CSG
XmNnavigationType	**XmCNavigationType**	**XmNavigation-Type**	XmNONE	G
XmNshadowThickness	**XmCShadowThickness**	**Dimension**	2	CSG
XmNtopShadowColor	**XmCTopShadowColor**	**Pixel**	dynamic	CSG
XmNtopShadowPixmap	**XmCTopShadowPixmap**	**Pixmap**	dynamic	CSG
XmNtraversalOn	**XmCTraversalOn**	**Boolean**	True	CSG
XmNunitType	**XmCUnitType**	**unsigned char**	dynamic	CSG
XmNuserData	**XmCUserData**	**Pointer**	NULL	CSG

Core Resource Set				
Name	**Class**	**Type**	**Default**	**Access**
XmNaccelerators	**XmCAccelerators**	**XtAccelerators**	NULL	CSG
XmNancestorSensitive	**XmCSensitive**	**Boolean**	dynamic	G
XmNbackground	**XmCBackground**	**Pixel**	dynamic	CSG
XmNbackgroundPixmap	**XmCPixmap**	**Pixmap**	XmUNSPECIFIED-_PIXMAP	CSG
XmNborderColor	**XmCBorderColor**	**Pixel**	XtDefaultForeground	CSG
XmNborderPixmap	**XmCPixmap**	**Pixmap**	XmUNSPECIFIED-_PIXMAP	CSG
XmNborderWidth	**XmCBorderWidth**	**Dimension**	0	CSG
XmNcolormap	**XmCColormap**	**Colormap**	dynamic	CG
XmNdepth	**XmCDepth**	**int**	dynamic	CG
XmNdestroyCallback	**XmCCallback**	**XtCallbackList**	NULL	C
XmNheight	**XmCHeight**	**Dimension**	dynamic	CSG
XmNinitial-ResourcesPersistent	**XmCInitial-ResourcesPersistent**	**Boolean**	True	CG
XmNmapped-WhenManaged	**XmCMapped-WhenManaged**	**Boolean**	True	CSG
XmNscreen	**XmCScreen**	**Screen** *	dynamic	CG
XmNsensitive	**XmCSensitive**	**Boolean**	True	CSG
XmNtranslations	**XmCTranslations**	**XtTranslations**	NULL	CSG
XmNwidth	**XmCWidth**	**Dimension**	dynamic	CSG
XmNx	**XmCPosition**	**Position**	0	CSG
XmNy	**XmCPosition**	**Position**	0	CSG

Callback Information

A pointer to the **DtTermStatusChangeCallbackStruct** callback structure, which includes at least the following members, is passed to callbacks for **DtNstatusChangeCallback**.

`int`	`reason`	Indicates why the callback was invoked: DtCR_TERM_STATUS_CHANGE.
`XEvent`	`*event`	Points to the **XEvent**, if any, that triggered the callback or NULL.
`int`	`cursorX`	The current text cursor X (column) position.
`int`	`cursorY`	The current text cursor Y (row) position.
`Boolean`	`capsLock`	The current state of the caps lock indicator.
`Boolean`	`stop`	The current state subprocess output parsing. Processing output from the subprocess can be turned on and off via the *stop*( ) action (similar to the XON / XOFF handshake invoked via <control>-S / <control>-Q).
`DtTermInsertCharMode`	`insertCharMode`	Not used by the DtTerm widget.
`Boolean`	`locked`	The current state of the keyboard caps lock.

A pointer to the **DtTermSubprocessTerminationCallbackStruct** callback structure, which includes at least the following members, is passed to callbacks for **DtNsubprocessTerminationCallback**.

`int`	`reason`	Indicates why the callback was invoked: DtCR_TERM_SUBPROCESS_TERMINATION.
`XEvent`	`*event`	Points to the **XEvent**, if any, that triggered the callback or NULL.
`pid_t`	`pid`	The process ID of the terminated subprocess.
`int`	`status`	The exit status of the terminated subprocess.

A pointer to the **DtTermInputVerifyCallbackStruct** callback structure, which includes at least the following members, is passed to callbacks for **DtNinputVerifyCallback**.

`int`	`reason`	Indicates why the callback was invoked: DtCR_TERM_INPUT_VERIFY.
`XEvent`	`*event`	Points to the **XEvent**, if any, that triggered the callback or NULL.
`Boolean`	`doit`	Indicates whether the text should be sent to the child process. Setting *doit* to False negates the action.
`unsigned char`	`*text`	Points to the text (either single- or multi-byte depending on the locale) to be sent to the child process.
`int`	`length`	Specifies the number of bytes to be sent to the child process.

A pointer to the **DtTermOutputlogVerifyCallbackStruct** callback structure, which includes at least the following members, is passed to callbacks for **DtNoutputLogCallback**.

`int`	`reason`	Indicates why the callback was invoked: DtCR_TERM_OUTPUT_LOG.
`XEvent`	`*event`	Points to the **XEvent**, if any, that triggered the callback or NULL.
`unsigned char`	`*text`	Points to the text (either single- or multi-byte depending on the locale) received from the child process.
`int`	`length`	Specifies the number of bytes received from the child process.

Translations

The DtTerm widget includes translations from *XmPrimitive*.

Altering translations in **#override** or **#augment** mode is undefined.

Key Pressed	Action Routine
`Shift ~Ctrl<Key>KP_Multiply:`	`XtDisplayInstalledAccelerators( )`
`~Shift Ctrl<Key>KP_Multiply:`	`XtDisplayAccelerators( )`
`Shift Ctrl<Key>KP_Multiply:`	`XtDisplayTranslations( )`
`<Key>osfCancel:`	`process-cancel( )`
`<Key>osfCopy:`	`copy-clipboard( )`
`<Key>osfCut:`	`copy-clipboard( )`
`<Key>osfPaste:`	`paste-clipboard( )`
`<Key>osfBeginLine:`	`beginning-of-buffer( )`
`<Key>osfEndLine:`	`end-of-buffer( )`
`Shift<Key>osfUp:`	`scroll(1,line)`
`Shift<Key>osfDown:`	`scroll(–1,line)`
`<Key>osfUp:`	`move-cursor(up)`
`<Key>osfDown:`	`move-cursor(down)`
`<Key>osfLeft:`	`move-cursor(backward)`
`<Key>osfRight:`	`move-cursor(forward)`
`<Key>Do:`	`vt-edit-key(do)`
`<Key>Help:`	`vt-edit-key(help)`
`<Key>Menu:`	`vt-edit-key(menu)`
`<Key>Find:`	`vt-edit-key(find)`
`<Key>Insert:`	`vt-edit-key(insert)`
`<Key>Select:`	`vt-edit-key(select)`
`~Shift<Key>osfPageUp:`	`vt-edit-key(prior)`
`~Shift<Key>osfPageDown:`	`vt-edit-key(next)`
`<Key>osfPageUp:`	`scroll(–1,page)`
`<Key>osfPageDown:`	`scroll(1,page)`
`Mod1<Key>Break:`	`soft-reset( )`
`Shift<Key>Break:`	`hard-reset( )`
`~Shift ~Mod1<Key>Break:`	`vt-break( )`
`Ctrl<Key>Cancel:`	`stop(long)`
`~Ctrl<Key>Cancel:`	`stop( )`
`~Shift<Key>Tab:`	`tab( )`

```
~Mod1<Key>KP_Space:          keypad-key-execute(space)
~Mod1<Key>KP_Tab:            keypad-key-execute(tab)
~Mod1<Key>KP_Enter:          keypad-key-execute(enter)
~Mod1<Key>KP_F1:             keypad-key-execute(f1)
~Mod1<Key>KP_F2:             keypad-key-execute(f2)
~Mod1<Key>KP_F3:             keypad-key-execute(f3)
~Mod1<Key>KP_F4:             keypad-key-execute(f4)
~Mod1<Key>KP_Equal:          keypad-key-execute(equal)
~Mod1<Key>KP_Multiply:       keypad-key-execute(multiply)
~Mod1<Key>KP_Add:            keypad-key-execute(add)
~Mod1<Key>KP_Separator:      keypad-key-execute(separator)
~Mod1<Key>KP_Subtract:       keypad-key-execute(subtract)
~Mod1<Key>KP_Decimal:        keypad-key-execute(decimal)
~Mod1<Key>KP_Divide:         keypad-key-execute(divide)
~Mod1<Key>KP_0:              keypad-key-execute(0)
~Mod1<Key>KP_1:              keypad-key-execute(1)
~Mod1<Key>KP_2:              keypad-key-execute(2)
~Mod1<Key>KP_3:              keypad-key-execute(3)
~Mod1<Key>KP_4:              keypad-key-execute(4)
~Mod1<Key>KP_5:              keypad-key-execute(5)
~Mod1<Key>KP_6:              keypad-key-execute(6)
~Mod1<Key>KP_7:              keypad-key-execute(7)
~Mod1<Key>KP_8:              keypad-key-execute(8)
~Mod1<Key>KP_9:              keypad-key-execute(9)
Shift<Key>F1:                vt-function-key-execute(1, UDK)
Shift<Key>F2:                vt-function-key-execute(2, UDK)
Shift<Key>F3:                vt-function-key-execute(3, UDK)
Shift<Key>F4:                vt-function-key-execute(4, UDK)
Shift<Key>F5:                vt-function-key-execute(5, UDK)
Shift<Key>F6:                vt-function-key-execute(6, UDK)
Shift<Key>F7:                vt-function-key-execute(7, UDK)
Shift<Key>F8:                vt-function-key-execute(8, UDK)
Shift<Key>F9:                vt-function-key-execute(9, UDK)
Shift<Key>F10:               vt-function-key-execute(10, UDK)
Shift<Key>F11:               vt-function-key-execute(11, UDK)
Shift<Key>F12:               vt-function-key-execute(12, UDK)
Shift<Key>F13:               vt-function-key-execute(13, UDK)
Shift<Key>F14:               vt-function-key-execute(14, UDK)
Shift<Key>F15:               vt-function-key-execute(15, UDK)
Shift<Key>F16:               vt-function-key-execute(16, UDK)
Shift<Key>F17:               vt-function-key-execute(17, UDK)
Shift<Key>F18:               vt-function-key-execute(18, UDK)
Shift<Key>F19:               vt-function-key-execute(19, UDK)
Shift<Key>F20:               vt-function-key-execute(20, UDK)
Shift<Key>F21:               vt-function-key-execute(21, UDK)
Shift<Key>F22:               vt-function-key-execute(22, UDK)
Shift<Key>F23:               vt-function-key-execute(23, UDK)
Shift<Key>F24:               vt-function-key-execute(24, UDK)
```

```
Shift<Key>F25:              vt-function-key-execute(25, UDK)
Shift<Key>F26:              vt-function-key-execute(26, UDK)
Shift<Key>F27:              vt-function-key-execute(27, UDK)
Shift<Key>F28:              vt-function-key-execute(28, UDK)
Shift<Key>F29:              vt-function-key-execute(29, UDK)
Shift<Key>F30:              vt-function-key-execute(30, UDK)
Shift<Key>F31:              vt-function-key-execute(31, UDK)
Shift<Key>F32:              vt-function-key-execute(32, UDK)
Shift<Key>F33:              vt-function-key-execute(33, UDK)
Shift<Key>F34:              vt-function-key-execute(34, UDK)
Shift<Key>F35:              vt-function-key-execute(35, UDK)
~Shift<Key>F1:              vt-function-key-execute(1, function)
~Shift<Key>F2:              vt-function-key-execute(2, function)
~Shift<Key>F3:              vt-function-key-execute(3, function)
~Shift<Key>F4:              vt-function-key-execute(4, function)
~Shift<Key>F5:              vt-function-key-execute(5, function)
~Shift<Key>F6:              vt-function-key-execute(6, function)
~Shift<Key>F7:              vt-function-key-execute(7, function)
~Shift<Key>F8:              vt-function-key-execute(8, function)
~Shift<Key>F9:              vt-function-key-execute(9, function)
~Shift<Key>F10:             vt-function-key-execute(10, function)
~Shift<Key>F11:             vt-function-key-execute(11, function)
~Shift<Key>F12:             vt-function-key-execute(12, function)
~Shift<Key>F13:             vt-function-key-execute(13, function)
~Shift<Key>F14:             vt-function-key-execute(14, function)
~Shift<Key>F15:             vt-function-key-execute(15, function)
~Shift<Key>F16:             vt-function-key-execute(16, function)
~Shift<Key>F17:             vt-function-key-execute(17, function)
~Shift<Key>F18:             vt-function-key-execute(18, function)
~Shift<Key>F19:             vt-function-key-execute(19, function)
~Shift<Key>F20:             vt-function-key-execute(20, function)
~Shift<Key>F21:             vt-function-key-execute(21, function)
~Shift<Key>F22:             vt-function-key-execute(22, function)
~Shift<Key>F23:             vt-function-key-execute(23, function)
~Shift<Key>F24:             vt-function-key-execute(24, function)
~Shift<Key>F25:             vt-function-key-execute(25, function)
~Shift<Key>F26:             vt-function-key-execute(26, function)
~Shift<Key>F27:             vt-function-key-execute(27, function)
~Shift<Key>F28:             vt-function-key-execute(28, function)
~Shift<Key>F29:             vt-function-key-execute(29, function)
~Shift<Key>F30:             vt-function-key-execute(30, function)
~Shift<Key>F31:             vt-function-key-execute(31, function)
~Shift<Key>F32:             vt-function-key-execute(32, function)
~Shift<Key>F33:             vt-function-key-execute(33, function)
~Shift<Key>F34:             vt-function-key-execute(34, function)
~Shift<Key>F35:             vt-function-key-execute(35, function)
<KeyRelease>:               key-release( )
<KeyPress>:                 insert( )
```

```
~Shift~Ctrl<Btn1Down>:        grab-focus( )
Shift~Ctrl<Btn1Down>:         extend-start( )
~Ctrl<Btn1Motion>:            select-adjust( )
~Ctrl<Btn1Up>:                extend-end( )
~Shift<Btn2Down>:             process-bdrag( )
~Shift<Btn2Up>:               copy-to( )
<EnterWindow>:                enter( )
<LeaveWindow>:                leave( )
<FocusIn>:                    focus-in( )
<FocusOut>:                   focus-out( )
```

Action Routines

The DtTerm widget supports the following action routines:

bell(**[***percentage***]**)
Rings the keyboard bell at the specified *percentage* above or below the base volume.

break( )
Sends an RS232 break signal to the child process.

cancel( )
Sends a **CAN** (cancel) character to the child process.

copy-clipboard( )
Copies current selection to the clipboard.

copy-to( )
Sends the primary selection to the subprocess.

do( ) Sends the escape sequence (see Section 12.7 on page 283) associated with the **Do** key to the child process.

edit-key(*string*)
Sends the escape sequence (see Section 12.7 on page 283) associated with the corresponding edit key to the child process. The interpretation of these keys is application-specific. Valid values for *string* are:

```
find
insert
next
prior
remove
select
```

extend-start( )
Starts the extension of the currently selected text. The amount of text selected depends on the number of mouse clicks (see *grab-focus*( )).

extend-end( )
Extends the current selection. The amount of text selected depends on the number of mouse clicks (see *grab-focus*( )).

function-key-execute(*num***[,** *type***]**)
Sends the escape sequence (see Section 12.7 on page 283) associated with the corresponding function key *num* to the child process. Valid values for *num* are 1 to 35, inclusive. If *type* is set to **function** (or not set at all), the escape sequence (see

Section 12.7 on page 283) associated with function key *num* is sent to the child process. If *type* is set to UDK, then the string associated with user defined key *num* is sent to the child process.

grab-focus( )
Performs one of the following depending on the number of multiple mouse clicks. One click deselects any selected text and sets the selection anchor at the pointer position; two clicks selects a word; three clicks selects a line of text; and four clicks selects all text.

hard-reset( )
Performs a hard reset on the terminal emulator.

help( )
Sends the escape sequence (see Section 12.7 on page 283) associated with the DEC VT220 Help key to the child process. The interpretation of this key is application-specific.

keymap(name)
Defines a new translation table whose resource name is named with the suffix **Keymap** (case is significant). The name **None** restores the original translation table.

keypad-key-execute(*string*)
Sends the escape sequence (see Section 12.7 on page 283) associated with the corresponding keypad key to the child process. The interpretation of these keys is application-specific. Valid values for *string* are:

```
f1 – f4
space
tab
enter
equal
multiply
add
separator
subtract
decimal
divide
0 – 9
```

move-cursor(*direction*)
Sends the escape sequence (see Section 12.7 on page 283) associated with the corresponding cursor motion to the child process. The interpretation of these keys is application-specific. Valid values for *direction* are:

```
up
down
backward
forward
```

paste-clipboard( )
Sends the contents of the clipboard to the subprocess.

process-bdrag( )
The result of this action is determined by several factors: position of the location cursor, motion of the location cursor and the interval between a **BTransfer** release.

This action sends the current selection to the subprocess if text is selected, the location cursor is disjoint from the current selection and no motion is detected within a given time interval.

The action drags the current selection if the location cursor is positioned on the selection, the time interval is exceeded and movement of the location cursor is detected. This action creates a **DragContext** object whose **XmNexportTargets** resource value includes target types of COMPOUND_TEXT, STRING and TEXT.

redraw-display( )
Redraws the contents of the text window.

scroll(*count***[,** *units***]**)
Scrolls the display memory down if *count* is greater than zero, or up if *count* is less than zero. The number of lines scrolled is based on count and units. The default for units is **line**. Valid values for *units* are:

```
page
halfpage
line
```

select-adjust( )
Extends the selection. The amount of text selected depends on the number of mouse clicks. One click selects characters; two clicks selects words; three clicks selects lines; and four clicks selects the entire buffer.

select-all( )
Selects all text.

select-page( )
Selects all text currently displayed on the screen.

self-insert( )
Sends the character associated with the key pressed to the child process.

soft-reset( )
Performs a soft reset of the terminal.

stop(*state*)
Toggles, starts, or stops the process of reading data from the child process. Valid values for *state* are:

```
toggle
on
off
```

string(*string*)
Inserts the specified text string as if it had been typed. The string must be quoted if it contains white space or non-alphanumeric characters. The string is interpreted as a hexadecimal character constant if it begins with the characters **0x**.

tab( ) Sends a tab to the child process.

visual-bell( )
Flashes the window quickly.

Virtual Bindings

The bindings for virtual keys are vendor-specific. Virtual bindings do not apply when the DtTerm widget has input focus. For information about bindings for virtual buttons and keys, see *VirtualBindings*( ).

SEE ALSO

dtterm, **<Dt/Term.h>**, *DtTermInitialize*( ), *DtTermDisplaySend*( ), *DtTermSubprocSend*( ), *DtTermSubprocReap*( ), *XtSetValues*( ) and *XtGetValues*( ), in the X / Open CAE Specification, **Window Management: X Toolkit Intrinsics**; *XmFrame*, *XmPrimitive*, *XmFontList*, *XmBulletinBoard*, *XmText*, in the X / Open CAE Specification, **Window Management: Xlib C Language Binding**; *VirtualBindings*, *Core* in the X / Open CAE Specification, **Motif Toolkit API**; *who* in the X / Open CAE Specification, **Commands and Utilities**, **Issue 4, Version 2**.

CHANGE HISTORY

First released in Issue 1.

12.4 Headers

This section describes the contents of headers used by the XCDE terminal emulation functions, macros and external variables.

Headers contain the definition of symbolic constants, common structures, preprocessor macros and defined types. Each function in Section 12.2 specifies the headers that an application must include in order to use that function. In most cases only one header is required. These headers are present on an application development system; they do not have to be present on the target execution system.

NAME

Dt/Term.h — terminal emulator definitions

SYNOPSIS

```
#include <Dt/Term.h>
```

DESCRIPTION

The **<Dt/Term.h>** header defines structures, values and function prototypes for terminal emulator services.

The header declares the following variable:

```
WidgetClass  dtTermWidgetClass;
```

The following are declared as functions:

```
Widget DtCreateTerm(Widget parent,
                    char *name,
                    Arg *arglist,
                    Cardinal argcount);

void DtTermInitialize(void);

void DtTermDisplaySend(Widget w,
                       unsigned char *buffer,
                       int length);

void DtTermSubprocSend(Widget w,
                       unsigned char *buffer,
                       int length);

void DtTermSubprocReap(pid_t pid,
                       int *stat_loc);
```

CHANGE HISTORY

First released in Issue 1.

12.5 Command-line Interfaces

This section defines the utilities that provide XCDE terminal emulation services.

NAME

dtterm — emulate a terminal window

SYNOPSIS

```
dtterm [±132] [±aw] [–background background_color] [–bd border_color]
[–bg background_color] [–bordercolor border_color]
[–borderwidth border_width] [±bs] [–bw border_width] [–C]
[–display display_name] [–e program_argument ...] [–fb fontset]
[–fg foreground_color] [–fn fontset] [–font fontset]
[–foreground foreground_color] [–geometry geometry_string] [–help]
[±iconic] [±j] [±kshMode] [±l] [–lf file_name] [±ls] [±map] [±mb]
[–ms pointer_color] [–name prog_name] [–nb number] [±rw] [–S ccn]
[–S c.n] [±sb] [±sf] [–sl screens[s | l]] [–ti term_id]
[–title title_string] [–tm term_modes] [–tn term_name] [–usage] [±vb]
[–w border_width] [–xrm resource_string]
```

DESCRIPTION

The *dtterm* utility provides runtime support of applications written for terminals conforming to the referenced ANSI X3.64-1979 standard and the referenced ISO/IEC 6429:1992 standard. The *dtterm* utility does not support the X/Open Utility Syntax Guidelines because it uses the X Window System convention of full-word options. The following options are available:

–132 Recognise the DECCOLM escape sequence and resize the window appropriately. Normally, *dtterm* ignores the DECCOLM escape sequence, which switches between 80- and 132-column mode.

+132 Ignore the DECCOLM escape sequence. This is the default behaviour.

–aw Allow auto-wraparound. This option allows the cursor to automatically wrap to the beginning of the next line when it is at the right-most position of a line and text is output. This is the default behaviour.

+aw Do not allow auto-wraparound.

–background *background_color*
Specify the terminal window background and the default background for the scroll bar and the X11 pointer cursor. This option defaults to either the primary colourset background (default) or select pixel (see **–bs**). The *background_color* argument describes the background colour.

–bd *border_color*
Specify the border colour for all windows. The shell widget's window border need not be visible when re-parenting window managers, such as the XCDE window manager, are used. The default is the colour black. The *border_color* argument describes the border colour.

–bg *background_color*
Equivalent to **–background**. The *background_color* argument describes the background colour.

–bordercolor *border_color*
Equivalent to **–bd**. The *border_color* argument describes the border colour.

–borderwidth *border_width*
Specify the border width of the shell widget's window. This value may be overridden by re-parenting window managers. The default is zero. The *border_width* argument specifies the width of the window border in pixels.

–**bs** Use the Motif select colour instead of the background colour for the terminal window's background colour.

+bs Do not use the Motif select colour instead of the background colour for the terminal window's background colour. This is the default behaviour.

–**bw** *border_width*
Equivalent to –**borderwidth**. The *border_width* argument specifies the width of the window border in pixels.

–**C** Specify that output directed to **/dev/console** be directed instead to the terminal window. It is provided as a way to prevent output, which would normally be displayed on the internal terminal emulator (ITE), from overwriting the X server's display. It is not provided as a general purpose mechanism to direct the output from an arbitrary system's **/dev/console** to an arbitrary X server. Ownership of, and read-write permission for, **/dev/console** is required in order to redirect console output.

–**display** *display_name*
Specify the X11 display server. This defaults to the *DISPLAY* environment variable. The *display_name* argument specifies the X11 display to which *dtterm* connects.

–**e** *program_argument ...*
Specify an executable program and any command-line arguments *dtterm* invokes as a subprocess when *dtterm* is started. It must be the last option on the command line. The *program_argument* arguments specify the program and any command-line arguments to be invoked by *dtterm*.

–**fb** *fontset*
Specify an **XFontSet** used by *dtterm* when displaying bold terminal text. The **XFontSet** should be specified as a Motif **XmFontList**. The terminal emulator supports only character or mono-spaced fonts. When using proportional fonts, the behaviour is undefined. The terminal emulator generates a default bold font based on the XLFD name of the **userFont**. If that font is not available, the terminal emulator generates bold text by overstriking (with a one pixel offset) the **userFont**. The *fontset* argument specifies the bold terminal **XFontSet** used by *dtterm*.

–**fg** *foreground_color*
Specify the foreground colour of the terminal window as well as the default foreground colour used by *dtterm* for the scroll bar and the for the X11 pointer cursor. This option defaults to either the primary colourset foreground (default) or select pixel. The *foreground_color* argument specifies the foreground colour.

–**fn** *fontset*
Specify an **XFontSet** used by *dtterm* when displaying terminal text. It should be specified as a Motif **XmFontList**. Only character or mono-spaced fonts are supported. When using proportional fonts, the behaviour is undefined. This font is not used to display non-terminal text (such as menu bar, popup menus or dialogs). The default uses the **XmNtextFontList** value of the parent bulletin board (see the XmBulletinBoard widget) in the same manner as the XmText widget. The *fontset* argument specifies the terminal **XFontSet**.

–**font** *fontset*
Equivalent to –**fn**. The *fontset* argument specifies the terminal **XFontSet**.

–**foreground** *foreground_color*
Equivalent to –**fg**. The *foreground_color* argument specifies the foreground colour used by *dtterm*.

–**geometry** *geometry_string*
Specify the terminal window's preferred size and position. Width and height are expressed in characters. The default size is 24 lines of 80 characters each. There is no default position. The *geometry_string* argument specifies the terminal geometry used by *dtterm*.

–**help** Display a message summarising *dtterm* usage.

–**iconic**
Display the terminal emulator initially in an iconified state.

+iconic
Display the terminal emulator initially as a normal window. This is the default behaviour.

–**j** Use jump scrolling. With jump scrolling, the screen may be scrolled more than one line at a time. This provides for faster screen updates when multiple lines of text are sent to the terminal. The maximum number of lines that may be jump scrolled is limited to the number of lines in the terminal window. The *dtterm* terminal emulator guarantees that all lines are displayed. This is the default behaviour.

+j Do not use jump scrolling.

–**kshMode**
Enable *ksh* mode. In *ksh* mode, a key pressed with the extend modifier bit set generates an escape character followed by the character generated by the un-extended keystroke. This option is provided for use with *emacs* command-line editor mode of *ksh*. It conflicts with the normal meta key use for generating extended single byte characters and for generating multi-byte Asian characters.

+kshMode
Do not enable *ksh* mode. This is the default behaviour.

–**l** Enables output logging. When *dtterm* enables logging, all output received from the subprocess is logged either to a file or to a command pipeline (as specified via the –**lf** option described in the following paragraph). Since data are logged directly from the subprocess, the log file includes all escape characters and carriage-return and newline pairs the terminal line discipline sends. The application may enable and disable logging via escape sequences.

+l Disable output logging. This is the default behaviour.

–**lf** *file_name*
Name the file to which *dtterm* writes the output log. If the *file_name* argument begins with a pipe symbol (|), *dtterm* assumes the rest of the string to be a command to be used as the endpoint of a pipe. The default file name is **DttermLog***XXXXX* (where *XXXXX* is a unique value) and is created in the directory from which the subprocess was started. The *file_name* argument specifies the log file name used by *dtterm* for logging.

–**ls** Start a login shell (the first character of *argv*[0] is a dash), indicating to the shell that it should read the system's **profile** and the user's **.profile** files (for *ksh* and *sh*).

+ls Start a normal (non-login) shell. This is the default behaviour.

–**map** Map (de-iconify) *dtterm* upon subprocess output if *dtterm* is unmapped (iconified). The user can specify, via the **mapOnOutputDelay** resource, an initial period of time during which *dtterm* does not map itself upon subprocess output.

+map Indicate there is no special mapping behaviour. This is the default behaviour.

–**mb** Ring a bell when the user types at a specified distance from the right margin. The distance from the right margin is specified by the –**nb** option.

+mb Do not ring a bell when the user types near the right margin. This is the default behaviour.

–**ms** *pointer_color*
Specify the foreground colour used by *dtterm* for the terminal window's (X11) pointer cursor. The default is the terminal window's foreground colour (see –**foreground**). The *pointer_color* argument specifies the pointer foreground colour used by *dtterm*.

–**name** *prog_name*
Specify the X11 name of the *dtterm* window. The *prog_name* argument specifies the name to use.

–**nb** *number*
Ring the bell this number of characters from the right margin when enabled. The default is 10. The *number* argument specifies the number of characters.

–**rw** Enable reverse-wraparound.

+rw Do not enable reverse-wraparound. This is the default behaviour.

–**S***ccn*
Run the terminal emulator against a pre-opened pseudo-terminal device. The terminal emulator provides this option to use when the pseudo-terminal device name is of the form **tty??** (that is, exactly two characters following the **tty**). This option is intended for use when *dtterm* is programmatically invoked from within another application. The *cc* argument specifies the last two characters of the pseudo-terminal device's slave name where the pseudo-terminal device slave name is of the form **tty??**. This value is ignored, but must be exactly two characters in length. The *n* argument specifies the file descriptor number that corresponds to the pseudo-terminal device's already opened master side.

–**S***c.n*
Equivalent to –**S***ccn*, but provided for systems with a larger pseudo-terminal device name space. The *c* argument specifies the last component of the pseudo-terminal device slave name. The terminal emulator ignores this value and the value may be empty. The *n* argument specifies the number of the file descriptor that corresponds to the pseudo-terminal device's already opened master side.

–**sb** Display a scroll bar. This is the default behaviour.

+sb Do not display a scroll bar.

–**sf** Generate Sun Function Key escape sequences instead of the escape sequences described in Section 12.7 on page 283 for the terminal's function keys. See Section 12.7.3 on page 296 for a description of the Sun Function Key escape sequences.

+sf Generate the escape sequences described in Section 12.7 on page 283 instead of Sun Function Key escape sequences for the terminal's function keys. This is the default behaviour.

–**sl** *screens*[**s**|**l**]
Specify the number of lines in the terminal buffer beyond the length of the window. The option value consists of a number followed by an optional suffix. If no suffix is included or the suffix is ''l'' (ell), the total length of the terminal buffer is *screens* plus the length of the terminal window. If the suffix is ''s'' (ess) the total length of the terminal buffer is (*screens*+1) times the length of the terminal window. The *dtterm* utility attempts to maintain the same buffer-to-window ratio when the window is resized larger. The default is **4s**. The *screens* argument specifies the number of screens or lines to save.

–**ti** *term_id*
Specify the name used by *dtterm* to select the correct response to terminal ID queries. Valid values are **vt100**, **vt101**, **vt102**, and **vt220**. The default is **vt220**. The *term_id* argument specifies the terminal ID to use.

–**title** *title_string*
Specify the window title. When used with the –**e** option, the default is the last component of the program's path; otherwise, the default is the last component of the name used to execute *dtterm* (that is, *argv*[0]). The *title_string* argument specifies the title used by *dterm*.

–**tm** *term_modes*
Specify a string containing terminal-setting keywords and the characters to which they can be bound. Allowable keywords include: **intr**, **quit**, **erase**, **kill**, **eof**, **eol**, **swtch**, **start**, **stop**, **brk**, **susp**, **dsusp**, **rprnt**, **flush**, **weras** and **lnext**. The terminal emulator correctly parses and silently ignores keywords that do not apply to a specific architecture. Control characters can be specified as **^***char* (for example, **^c** or **^u**), and **^?** can be used to indicate Delete. The default is NULL. The *term_modes* argument specifies the terminal mode string.

–**tn** *term_name*
Specify a name to which *dtterm* sets the *TERM* environment variable. The default is *dtterm*. The *term_name* argument specifies the terminal name used by *dtterm*.

–**usage**
Display a usage message on the screen.

–**vb** Use a visual bell instead of an audible one. Flash the window instead of ringing the terminal bell whenever a <control>-G is received.

+vb Use an audio bell instead of a visual one. This is the default behaviour.

–**w** *border_width*
Equivalent to –**borderwidth**. The *border_width* argument specifies the width of the window border in pixels.

–**xrm** *resource_string*
Allow the user to specify the X11 Resource Manager-style resources on the command line. The *resource_string* argument specifies an X11 resource string. (See *XrmParseCommand*( ) and *XGetDefault*( ) for more information.)

OPERANDS

None.

STDIN

Not used.

INPUT FILES

None.

ENVIRONMENT VARIABLES

The following environment variables affect the execution of *dtterm*:

DISPLAY Specify the default X Windows display to connect to (see –**display**). The terminal emulator sets the subprocess's *DISPLAY* environment variable to the connected X11 display name.

HOME Determine the user's home directory, the location of configuration files.

LANG Provide a default value for the internationalisation variables that are unset or null. If *LANG* is unset or null, the corresponding value from the implementation-specific default locale will be used. If any of the internationalisation variables contains an invalid setting, the utility behaves as if none of the variables had been defined.

LC_ALL If set to a non-empty string value, override the values of all the other internationalisation variables.

LC_MESSAGES Determine the locale that is used to affect the format and contents of diagnostic messages written to standard error and informative messages written to standard output.

NLSPATH Determine the location of message catalogues for the processing of *LC_MESSAGES*.

SHELL Determine the default application to run.

XAPPLRESDIR Specify the name of a directory that contains application-specific resources. If this environment variable is defined, and is set to an existing directory, then it is searched (in addition to the standard locations) for files containing application-specific resource specifications.

XENVIRONMENT
Specify the name of a resource file with user- or machine-specific resources. If this variable is not defined, *dtterm* looks for a file named **$HOME/.Xdefaults-***hostname* instead, where *hostname* is the name of the host where the application is executing.

XFILESEARCHPATH
Define a language-dependent location of **app-defaults**.

XMODIFIER Specify which input method to use.

XUSERFILESEARCHPATH
Control where X applications look for their **app-defaults** resource files. The default is located in the directory **/usr/dt/app-defaults**. The user must set *XUSERFILESEARCHPATH* if the user's resource files are not in this location.

The terminal emulator creates the following variable when it invokes another process:

TERM The terminal emulator sets the subprocess's *TERM* environment variable to the **termName** resource value. Applications use this variable to determine the type of escape sequences to use when driving the terminal emulator.

TERMINAL_EMULATOR
The terminal emulator sets the subprocess's *TERMINAL_EMULATOR* environment variable to *dtterm* to indicate that the process is running from a *dtterm* terminal emulator.

WINDOWID The terminal emulator sets the subprocess's *WINDOWID* environment variable to the window number of the window in which text is rendered.

RESOURCES

The *dtterm* utility allows the user to specify behaviour through X11 resources as well as the command-line interface. The following is a list of the defined resources:

The dtterm Client Resource Set

Name	Class	Type	Default
allowSendEvents	**AllowSendEvents**	**Boolean**	False
appCursorDefault	**AppCursorDefault**	**Boolean**	False
appKeypadDefault	**AppKeypadDefault**	**Boolean**	False
autoWrap	**AutoWrap**	**Boolean**	True
background	**Background**	**String**	
backgroundIsSelect	**BackgroundIsSelect**	**Boolean**	False
blinkRate	**BlinkRate**	**int**	250
borderColor	**BorderColor**	**String**	‘‘black’’
borderWidth	**BorderWidth**	**int**	0
c132	**C132**	**Boolean**	False
charCursorStyle	**CharCursorStyle**	**String**	‘‘char_cursor_box’’
consoleMode	**ConsoleMode**	**Boolean**	False
foreground	**Foreground**	**String**	
geometry	**Geometry**	**String**	NULL
iconic	**Iconic**	**Boolean**	False
iconName	**IconName**	**String**	‘‘dtterm’’
jumpScroll	**JumpScroll**	**Boolean**	True
kshMode	**KshMode**	**Boolean**	False
logging	**Logging**	**Boolean**	False
logFile	**LogFile**	**String**	‘‘DttermLog*XXXXX*’’ (where *XXXXX* is a unique value)
logInhibit	**LogInhibit**	**Boolean**	False
loginShell	**LoginShell**	**Boolean**	False
mapOnOutput	**AutoMap**	**Boolean**	False
mapOnOutputDelay	**MapDelay**	**int**	0
marginBell	**MarginBell**	**Boolean**	False
menuBar	**MenuBar**	**Boolean**	True
menuPopup	**MenuPopup**	**Boolean**	True
nMarginBell	**NMarginBell**	**int**	10
pointerBlank	**PointerBlank**	**Boolean**	False
pointerBlankDelay	**PointerBlankDelay**	**int**	2
pointerColor	**Foreground**	**String**	foreground color
pointerColorBackground	**Background**	**String**	background color
pointerShape	**PointerShape**	**String**	‘‘xterm’’
reverseWrap	**ReverseWrap**	**Boolean**	False
saveLines	**SaveLines**	**String**	4s
scrollBar	**ScrollBar**	**Boolean**	True
sunFunctionKeys	**SunFunctionKeys**	**Boolean**	False
termId	**TermId**	**String**	‘‘vt220’’
termName	**TermName**	**String**	‘‘dtterm’’
title	**Title**	**String**	‘‘dtterm’’
ttyModes	**TtyModes**	**String**	NULL
userBoldFont	**UserBoldFont**	**XmFontList**	dynamic
userFont	**UserFont**	**XmFontList**	dynamic
visualBell	**VisualBell**	**Boolean**	False

allowSendEvents

Specifies that the terminal emulator allow synthetic events (generated and sent by another application). Enabling this resource opens up a possible security hole.

appCursorDefault
If True, the cursor keys are initially in application mode. If False, the cursor keys are initially in cursor mode.

appKeypadDefault
If True, the keypad keys are initially in application mode. If False, the keypad keys are initially in numeric mode.

autoWrap
Specifies whether or not auto-wraparound is initially enabled.

background
Specifies the background colour of the terminal window as well as the default background colour for the scroll bar. This resource defaults to either the primary colourset background or select pixel (see **backgroundIsSelect**).

backgroundIsSelect
Specifies that the terminal window should use the Motif select colour instead of the background colour for the terminal window's background colour.

blinkRate
Specifies the number of milliseconds the cursor is in the on and off states while blinking. A value of 250 blinks the cursor two times per second. A value of zero turns blinking off.

borderColor
Specifies the border colour for the window. The window border need not be visible when re-parenting window managers are used.

borderWidth
Specifies the border width of the shell widget's window. This value may be overridden by re-parenting window managers.

c132 Specifies whether or not the DECCOLM escape sequence, which switches between a 132- and 80-column window, is honoured.

charCursorStyle
This resource specifies the shape of the text cursor. A char_cursor_box value specifies a cursor the width and height of the base font's bounding box. A char_cursor_bar value specifies a cursor the width of the base font's bounding box, 2 pixels high, and drawn with its top on the baseline.

consoleMode
Specifies that output directed at **/dev/console** be directed instead to the terminal window. It is provided as a way to prevent output, that would normally be displayed on the internal terminal emulator (ITE), from overwriting the X server's display. It is not provided as a general mechanism to direct the output from an arbitrary system's **/dev/console** to an arbitrary X server. Ownership of, and read-write permission for, **/dev/console** is required in order to redirect console output.

foreground
Specifies the foreground of the terminal window as well as the default used by *dtterm* for the scroll bar and the colour used for the pointer cursor. This resource defaults to the primary colourset foreground pixel.

geometry
Specifies the terminal window's preferred size and position. The default size is 24 lines of 80 characters each. There is no default position.

iconGeometry
Specifies the preferred position of the terminal emulator's icon. Window managers may ignore this value. There is no default.

iconic
Specifies whether or not the terminal emulator is initially displayed in an iconified state.

iconName
Specifies the name for the icon. When used with the **–e** option, the default is the last component of the program's path; otherwise, the default is the last component of the name used to execute *dtterm* (that is, *argv*[0]).

jumpScroll
Specifies that *dtterm* use jump scrolling. With jump scrolling, the screen may be scrolled more than one line at a time. This provides for faster screen updates when multiple lines of text are sent to the terminal. The maximum number of lines that may be jump scrolled is limited to the number of lines in the terminal window. The *dtterm* terminal emulator guarantees that all lines are displayed.

kshMode
Enables ksh mode. With ksh mode, a key pressed with the extend modifier bit set generates an escape character followed by the character generated by the un-extended keystroke. This option is provided for use with the *emacs* command-line editor mode of *ksh*. It conflicts with the normal meta key use for generating extended single byte characters and for generating multi-byte Asian characters.

logging
Enables output logging. When logging is enabled, all output received from the subprocess is logged either to a file or to a command pipeline (as specified via the **logFile** option). Since the data is logged directly from the subprocess, it includes all escape characters and carriage-returns and newline pairs the terminal line discipline sends. Logging may be enabled and disabled via escape sequences.

logFile
Specifies the filename to which *dtterm* writes the output log. If the filename begins with a pipe symbol (|), *dtterm* assumes the rest of the string is a command to be used as the endpoint of a pipe. The default filename is **DttermLog***XXXXX* (where *XXXXX* is a unique value) and is created in the directory from which the subprocess was started.

logInhibit
Indicates that *dtterm* inhibit device and file logging.

loginShell
Indicates that the shell that is started be a login shell (that is, the first character of *argv*[0] is a dash), indicating that the shell should read the system's **profile** and the user's **.profile** files (for *ksh* and *sh*).

mapOnOutput
Indicates that the terminal emulator map (de-iconify) itself upon subprocess output if it is unmapped (iconified). The user can specify, via the **mapOnOutputDelay** resource, an initial period of time during which *dtterm* does not map itself upon subprocess output.

mapOnOutputDelay
Specifies the number of seconds after start-up that *dtterm* does not honour the

mapOnOutput resource. This allows the application to send initial output (for example, shell prompts) to the terminal without auto mapping the window. The default is zero (no delay).

marginBell
Specifies whether or not the bell rings when the user types near the right margin. The distance from the right margin is specified by the **nMarginBell** resource.

menuBar
Indicates that *dtterm* displays a pulldown menu bar. The default is True.

menuPopup
Indicates that *dtterm* displays a popup menu. The default is True.

nMarginBell
Specifies the number of characters from the right margin at which the margin bell rings, when enabled.

pointerBlank
Specifies that *dtterm* puts the pointer cursor into blanking mode. In this mode, the cursor turns on when the pointer is moved, and is blanked after a selectable number of seconds or after keyboard input. The **pointerBlankDelay** resource sets the delay.

pointerBlankDelay
Specifies the number of seconds to wait after the pointer has stopped moving before blanking the pointer (see **pointerBlank**). A value of zero delays pointer blanking until a key is pressed.

pointerColor
Specifies the foreground colour used by *dtterm* for the terminal window's pointer (X11) cursor. The default is the terminal window's foreground colour (see **foreground**).

pointerColorBackground
Specifies the background colour used by *dtterm* for the terminal window's pointer (X11) cursor. The default is the terminal window's background colour (see **background**).

pointerShape
Specifies the X cursor font character used by *dtterm* as the pointer cursor. The font character must be specified as a string from the **<X11/cursorfont.h>** header with the leading **XC_** removed. The default is *xterm*.

reverseWrap
Specifies whether or not reverse-wraparound is enabled.

saveLines
Specifies the number of lines in the terminal buffer beyond the length of the window. The option value consists of a number followed by an optional suffix. If no suffix is included or the suffix is ''l'' (ell), the total length of the terminal buffer is *screens* plus the length of the terminal window. If the suffix is ''s'' (ess) the total length of the terminal buffer is (*screens*+1) times the length of the terminal window. The *dtterm* utility attempts to maintain the same buffer-to-window ratio when the window is resized larger.

scrollBar
Specifies that *dtterm* displays a scroll bar.

sunFunctionKeys
Specifies whether *dtterm* generates Sun Function Key escape sequences instead of the escape sequences described in Section 12.7 on page 283 for the terminal's function keys. See Section 12.7.3 on page 296 for a description of the Sun Function Key escape sequences.

termId
Supplies the name used to select the correct response to terminal ID queries. Valid values are **vt100**, **vt101**, **vt102**, and **vt220**.

termName
Specifies a name to which *dtterm* sets the *TERM* environment variable. The default is *dtterm*.

title
Specifies the window title. When used with the **–e** option, the default is the last component of the program's path; otherwise, the default is the last component of the name used to execute *dtterm* (that is, *argv*[0]).

ttyModes
Specifies a string containing terminal-setting keywords and the characters to which they can be bound. Allowable keywords include: **intr**, **quit**, **erase**, **kill**, **eof**, **eol**, **swtch**, **start**, **stop**, **brk**, **susp**, **dsusp**, **rprnt**, **flush**, **weras** and **lnext**. The terminal emulator correctly parses and silently ignores keywords that do not apply to a specific architecture. Control characters can be specified as **^***char* (for example, **^c** or **^u**), and **^?** can be used to indicate Delete.

userBoldFont
Specifies an **XFontSet** used by *dtterm* when displaying bold terminal text. The **XFontSet** should be specified as a Motif **XmFontList**. The terminal emulator supports only character or mono-spaced fonts. When using proportional fonts, the behaviour is undefined. The terminal emulator generates a default bold font based on the **XLFD** name of the **userFont**. If that font is not available, *dtterm* generates bold text by overstriking (with a one pixel offset) the **userFont**.

userFont
Specifies an **XFontSet** used by *dtterm* when displaying terminal text. **XFontSet** should be specified as a Motif **XmFontList**. The terminal emulator supports only character or mono-spaced fonts. When using proportional fonts, the behaviour is undefined. This font is not used to display non-terminal text (such as menu bar, popup menu and dialog). The default is the **XmNtextFontList** value of the parent bulletin board (see the XmBulletinBoard widget) in the same manner as the XmText widget.

visualBell
Indicates that a visual bell is preferred over an audible one. Instead of ringing the terminal bell whenever a <control>-G is received, the window is flashed.

Resource/Option Correspondence

Many of the preceding resources correspond to the command-line arguments. The following table describes the relationship between the two:

Command-line option	Resource Setting
–132	∗**c132: True**
+132	∗**c132: False**
–aw	∗**autoWrap: True**
+aw	∗**autoWrap: False**
–background *background_color*	∗**background:** *background_color*
–bd *border_color*	∗**borderColor:** *border_color*
–bg *background_color*	∗**background:** *background_color*
–bordercolor *border_color*	∗**borderColor:** *border_color*
–borderwidth *border_width*	**.borderWidth:** *border_width*
–bs	∗**backgroundIsSelect: True**
+bs	∗**backgroundIsSelect: False**
–bw *border_width*	**.borderWidth:** *border_width*
–C	∗**consoleMode: True**
–display *display_name*	**.display:** *display_name*
–e *program_argument...*	
–fb *fontset*	∗**userBoldFont:** *fontset*
–fg *foreground_color*	∗**foreground:** *foreground_color*
–fn *fontset*	∗**userFont:** *fontset*
–font *fontset*	∗**userFont:** *fontset*
–foreground *foreground_color*	∗**foreground:** *foreground_color*
–geometry *geometry_string*	**.geometry:** *geometry_string*
–iconic	**.iconic: True**
+iconic	**.iconic: False**
–j	∗**jumpScroll: True**
+j	∗**jumpScroll: False**
–kshMode	∗**kshMode: True**
+kshMode	∗**kshMode: False**
–l	∗**logging: True**
+l	∗**logging: False**
–lf *file_name*	∗**logFile:** *file_name*
–ls	∗**loginShell: True**
+ls	∗**loginShell: False**
–map	∗**mapOnOutput: True**
+map	∗**mapOnOutput: False**
–mb	∗**marginBell: True**
+mb	∗**marginBell: False**
–ms *pointer_color*	∗**pointerColor:** *pointer_color*
–name *prog_name*	**.name:** *prog_name*
–nb *number*	∗**nMarginBell:** *number*
–rw	∗**reverseWrap: True**
+rw	∗**reverseWrap: False**
–sb	∗**scrollBar: True**
+sb	∗**scrollBar: False**

−sf	∗**sunFunctionKeys: True**
+sf	∗**sunFunctionKeys: False**
−sl *screens***s**	∗**saveLines:** *screens* ∗ **lines/screen**
−sl *lines*	∗**saveLines:** *lines*
−ti *term_id*	∗**termId:** *term_id*
−title *title_string*	**.title:** *title_string*
−tm *term_modes*	∗**ttyModes:** *term_modes*
−tn *term_name*	∗**termName:** *term_name*
−vb	∗**visualBell: True**
+vb	∗**visualBell: False**
−w *border_width*	**.borderWidth:** *border_width*

ASYNCHRONOUS EVENTS
Default.

STDOUT
Not used.

STDERR
Used only for diagnostic messages.

OUTPUT FILES
None.

EXTENDED DESCRIPTION
None.

EXIT STATUS

0 successful completion

>0 an error occurred

CONSEQUENCES OF ERRORS
Default.

APPLICATION USAGE
None.

EXAMPLES
None.

SEE ALSO
<Dt/Term>, *XmBulletinBoard*, *XmText* in the X/Open CAE Specification, **Motif Toolkit API**; *XrmParseCommand*( ), *XGetDefault*( ) in the X/Open CAE Specification, **Window Management: Xlib C Language Binding**.

CHANGE HISTORY
First released in Issue 1.

12.6 Actions

This section defines the actions that provide XCDE terminal emulation services to support application portability at the C-language source or shell script levels.

NAME

dttermaction — XCDE terminal emulation actions

SYNOPSIS

```
Dtterm
Terminal
```

DESCRIPTION

The XCDE Terminal Emulation Services support the following terminal emulation actions:

Dtterm
Open a view of the desktop terminal emulator.

Terminal
Open a view of the user's preferred terminal emulator.

These actions can be invoked from an application using the *DtActionInvoke*( ) function or invoked from a command line using the *dtaction* utility.

SEE ALSO

dtaction, *DtActionInvoke*( ) in the **XCDI** specification.

CHANGE HISTORY

First released in Issue 1.

12.7 Formats

12.7.1 Received Escape Sequences

The *dtterm* utility and the DtTerm widget support the following list of received escape sequences. Spaces have been added for readability and are not part of the escape sequence. The following indicate parameters: *pi*, *p1*, *label*, *file* and *text*. *Space* indicates a required space, hexadecimal code 0x20. A <control>-*char* indicates a control code (such as <control>-G, which is hexadecimal code 0x07). *Esc* indicates hexadecimal code 0x1b. *Backslash* indicates hexadecimal code 0x5c. Literals are indicated as **literal** and must be included exactly as specified. All references to the *dtterm* utility in this section also apply to the DtTerm widget.

<control>-G
: (BEL) Bell. The terminal either issues an audible bell, or flashes the text window depending on the state of the visual bell flag.

<control>-H
: (BS) Backspace. The cursor moves one cursor position to the left. If reverse-wrap mode is disabled and the cursor is at the left-most column of the line when a backspace character is received, the cursor remains at its current position. If reverse-wrap mode is enabled and the cursor is at the left-most column of the line when a backspace character is received, the cursor moves to the right-most column of the previous line. If the cursor is also in the top-most row, the cursor moves to the right-most column of the bottom-most row.

<control>-I
: (HT) Horizontal Tab. The cursor moves right to the next tab stop. If there are no further tab stops set to the right of the cursor, the cursor moves to the right-most column of the current line.

<control>-J
: (LF) Line Feed or New Line. The cursor moves to the same column of the next line. If the cursor is in the bottom-most line of the scrolling region, the scrolling region scrolls up one line. Lines scrolled off the top of the scrolling region are lost. Blank lines with no visible character attributes are added at the bottom of the scrolling region.

<control>-K
: (VT) Vertical Tab. Same as Line Feed.

<control>-L
: (FF) Form Feed or New Page. Same as Line Feed.

<control>-M
: (CR) Carriage Return. The cursor moves to the left-most column of the current line.

Esc **(B**
: (SCS) Designate referenced ISO / IEC 646: 1983 standard (base font) as G0.

Esc **(0**
: (SCS) Designate DEC Special Graphic (line draw) as G0.

Esc **) B**
: (SCS) Designate referenced ISO / IEC 646: 1983 standard (base font) as G1.

Esc **) 0**
: (SCS) Designate DEC Special Graphic (line draw) as G1.

Esc * **B**
(SCS) Designate referenced ISO / IEC 646: 1983 standard (base font) as G2.

Esc * **0**
(SCS) Designate DEC Special Graphic (line draw) as G2.

Esc **+ B**
(SCS) Designate referenced ISO / IEC 646: 1983 standard (base font) as G3.

Esc **+ 0**
(SCS) Designate DEC Special Graphic (line draw) as G3.

<control>-N
(LS1) Map G1 into GL.

<control>-O
(LS0) Map G0 into GL.

Esc **n** (LS2) Map G2 into GL.

Esc **o** (LS3) Map G3 into GL.

Esc **N** (SS2) Map G2 into GL for the next character.

Esc **O** (SS3) Map G3 into GL for the next character.

Esc Space **F**
(S7C1T) Select 7-bit C1 Control Characters. In this mode, the **dtterm** utility sends all C1 Control Characters to the host as 7-bit escape sequences. That is, CSI is sent to the host as ''*Esc* ['' .

Esc Space **G**
(C8C1T) Select 8-bit C1 Control Characters. In this mode, the *dtterm* utility sends all C1 Control Characters to the host as 8-bit control codes. That is, CSI is sent back as the hexadecimal value 0x9B.

Esc **# 8**
(DECALN) DEC Screen Align Test. The screen is filled with the character ''E''.

Esc **7** (DECSC) Save cursor. The following is saved:

- Cursor position
- Character attributes set by the SGR command
- Any pending single shift 2 or 3 (SS2 or SS3)
- State of the autowrap flag
- State of the reverse wrap flag
- State of origin mode (DECOM)
- State of selective erase

Esc **8** (DECRC) Restore cursor. The terminal emulator is restored to the state saved by the save cursor (DECSC) function. If nothing was saved by DECSC, then the following actions are performed:

- Moves the cursor to the home position
- Resets the origin mode (DECOM)

- Turns off all character attributes (SGR)
- Maps the referenced ISO / IEC 646: 1983 standard character set into GL

Esc **=** (DECPAM) Application keypad. In this mode, the numeric keypad sends application sequences. (See Section 12.7.3 on page 294).

Esc **>** (DECPNM) Normal keypad. In this mode, the numeric keypad sends the characters shown on the keypad. Keys PF1 to PF4, inclusive, send application sequences. (See Section 12.7.3 on page 294).

Esc **D** (IND) Index. The cursor moves down to the same column of the next line. If the cursor is in the bottom-most line of the scrolling region, the scrolling region is scrolled up one line. The line scrolled off the top of the scrolling region is lost. A blank line with no visible character attributes is added at the bottom of the scrolling region.

Esc **E** (NEL) Next line. The cursor moves down to the first column of the next line. If the cursor is in the bottom-most line of the scrolling region, the scrolling region is scrolled up one line. The line scrolled off the top of the scrolling region is lost. A blank line with no visible character attributes is added at the bottom of the scrolling region.

Esc **H** (HTS) Tab set. This function sets a horizontal tab stop at the column where the cursor is located.

Esc **M** (RI) Reverse index. The cursor moves up to the same column of the previous line. If the cursor is in the top-most line of the scrolling region, the scrolling region is scrolled down one line. The line scrolled off the bottom of the scrolling region is lost. A blank line with no visible character attributes is added at the top of the scrolling region.

Esc **P** *p1* **;** *p2* **|** *p3 Esc Backslash*
(DECUDK) User defined keys

Esc **Z** (DECID) Return terminal ID. This function is similar to a primary device attributes (DA) request. (See ‘‘*Esc* **[c** ’’ (DA) described later in this document.)

Esc **c** (RIS) Full reset. This function performs a full (hard) reset. For additional information, see Section 12.7.2 on page 293.

Esc **[** *pi* **q**
(DECSCA) Select character protection attribute. The default value is 0. This escape sequence defines the characters that come after it as erasable or not erasable from the screen. The selective erase escape sequences, (DECSED and DECSEL), can only erase characters defined as erasable. Valid supported values of *pi* are:

0 DECSED and DECSEL can erase characters.

1 DECSED and DECSEL cannot erase characters.

2 Same as **0**.

Esc **[** *pi* **@**
(ICH) Insert *pi* blank characters. The default value is 1. A parameter value of 0 or 1 inserts a single blank character. A parameter value of *N* inserts *N* blank characters. Blank characters with normal character attributes are inserted at the cursor position. Characters to the right of the cursor move to the right. Characters scrolled past the end of the line are lost.

Esc **[** *pi* **A**

(CUU) Cursor up *pi* lines. The default value is 1. A parameter value 0 or 1 moves the cursor up one line. A parameter value of *N* moves the cursor up *N* lines. The cursor stops at the top margin. If the cursor is already above the top margin, the cursor stops at the top line.

Esc **[** *pi* **B**

(CUD) Cursor down *pi* lines. The default value is 1. A parameter value 0 or 1 moves the cursor down one line. A parameter value of *N* moves the cursor down *N* lines. The cursor stops at the bottom margin. If the cursor is already below the bottom margin, the cursor stops at the bottom line.

Esc **[** *pi* **C**

(CUF) Cursor forward *pi* characters. The default value is 1. A parameter value 0 or 1 moves the cursor forward one character. A parameter value of *N* moves the cursor forward *N* characters. The cursor stops at the right-most column of the line.

Esc **[** *pi* **D**

(CUB) Cursor backward *pi* characters. The default value is 1. A parameter value 0 or 1 moves the cursor backward one character. A parameter value of *N* moves the cursor backward *N* characters. The cursor stops at the left-most column of the line.

Esc **[** *pi* **F**

(CPL) Cursor to the first column of the *pi*th preceding *line*. The default value is 1. A parameter value 0 or 1 moves the cursor to the preceding line. A parameter value of *N* moves the cursor to the *N*th preceding line. If the cursor is below the top margin, the cursor stops at the top margin. If the cursor is already above the top margin, the cursor stops at the top line.

Esc **[** *pi* **G**

(CHA) Cursor to column *pi*. The default value is 1. A parameter value 0 or 1 moves the cursor to the first column of the current line. A parameter value of *N* moves the cursor to the *N*th column of the current line.

Esc **[** *p1* **;** *p2* **H**

(CUP) Cursor position. The default value is 1. A *p1* value 0 or 1 moves the cursor to row one. A *p1* value of *N* moves the cursor to row N. A *p2* value 0 or 1 moves the cursor to column one. A *p2* value of *N* moves the cursor to column N. The starting point for lines and columns depends on the setting of the origin mode (DECOM).

Esc **[** *pi* **J**

(ED) Erase in display. The default value is 0. A parameter value of 0 erases from the cursor to the end of the display. A parameter value of 1 erases from the beginning of the display to the cursor position, inclusive. A parameter value of 2 erases the complete display.

Esc **[** *pi* **K**

(EL) Erase in line. The default value is 0. A parameter value of 0 erases from the cursor to the end of the line. A parameter value of 1 erases from the beginning of the line to the cursor position, inclusive. A parameter value of 2 erases the complete line.

Esc **[** *pi* **L**

(IL) Insert lines. The default value is 1. A parameter value 0 or 1 inserts one line at the cursor. A parameter value of *N* inserts *N* lines at the cursor. As lines are inserted, lines below the cursor and in the scrolling region move down. Lines scrolled off the page are lost. There is no effect outside the scrolling region.

Esc **[** *pi* **M**

(DL) Delete lines. The default value is 1. A parameter value 0 or 1 deletes one line at the cursor. A parameter value of *N* deletes *N* lines at the cursor. As lines are deleted, lines below the cursor and in the scrolling region move up. Blank lines with no visible character attributes are added at the bottom of the scrolling region. There is no effect outside the scrolling region.

Esc **[** *pi* **P**

(DCH) Delete characters. The default value is 1. A parameter value 0 or 1 deletes one character at the cursor position. A parameter value of *N* deletes *N* characters at the cursor position. An parameter greater than the number of characters between the cursor and the right margin only deletes the remaining characters on the line. As characters are deleted, the remaining characters move left and are replaced by blank spaces with no visual character attributes.

Esc **[** *pi* **S**

(SU) Scroll up *pi* lines. The default value is 1. A parameter value 0 or 1 scrolls the display up one line. A parameter value of *N* scrolls the display up *N* lines. The scrolling region scrolls up. Lines scrolled off the top of the scrolling region are lost. Blank lines with no visible character attributes are added at the bottom of the scrolling region.

Esc **[** *pi* **T**

(SD) Scroll down *pi* lines. The default value is 1. A parameter value 0 or 1 scrolls the display down one line. A parameter value of *N* scrolls the display down *N* lines. The scrolling region scrolls down. Lines scrolled off the bottom of the scrolling region are lost. Blank lines with no visible character attributes are added at the top of the scrolling region.

Esc **[** *pi* **X**

(ECH) Erase *pi* characters. The default value is 1. A parameter value 0 or 1 erases a single character. A parameter value of *N* erases *N* characters. The character attributes of erased characters are cleared. This escape sequences works inside or outside the scrolling margins.

Esc **[** *pi* **c**

(DA) Send device attributes. The default is 0. A parameter value 0 or 1 causes the terminal emulator to respond with ''*Esc* **[? 1 ; 2 c**''.

Esc **[** *p1* **;** *p2* **f**

(HVP) Horizontal and vertical position. This escape sequence has been replaced by CUP and offers identical functionality. It is provided to maintain backward compatibility.

Esc **[** *pi* **g**

(TBC) Tab clear. The default is 0. A parameter value of 0 clears the tab stop at the current cursor column. A parameter value of 3 clears all tab stops.

Esc **[** *pi* **h**

(SM) Set mode. This escape sequence sets ANSI modes. Valid supported values of *pi* are:

2 (KAM) Keyboard lock. In this mode, *dtterm* ignores all keystrokes from the keyboard.

4 (IRM) Insert mode. In this mode, new characters move characters in display memory to the right. Characters moved past the end of the line are lost.

12 (SRM) Local echo off. In this mode, *dtterm* sends keyboard characters to the host only. The host must echo back characters for them to be displayed.

20 (LNM) New line. In this mode, the cursor moves to the first column on the next line when *dtterm* receives an LF, FF or VT character. When the Return key is pressed, *dtterm* sends a carriage-return (CR) followed by a newline (NL).

Esc **[** *pi* **l**

(RM) Reset mode. This escape sequences resets ANSI modes. Valid supported values of *pi* are:

2 (KAM) Keyboard unlock. In this mode, *dtterm* processes all keystrokes from the keyboard.

4 (IRM) Replace mode. In this mode, new characters replace the character at the cursor position.

12 (SRM) Local echo on. In this mode, *dtterm* sends keyboard characters to both the host and the display. The host does not have to echo back characters for them to be displayed.

20 (LNM) New line. In this mode, the cursor moves to the same column on the next line when *dtterm* receives an LF, FF or VT character. When the Return key is pressed, *dtterm* sends a carriage-return (CR).

Esc **[** *pi* **; ... m**

(SG) Graphics rendition. The default value is 0. This escape sequence selects one or more character attributes. Valid supported values for *pi* are:

0 All attributes off

1 Bold

2 Faint

4 Underline

5 Blinking. This attribute appears as bold text

7 Negative image

8 Invisible image

22 Bold and Faint off

24 Underline off

25 Blinking off

27 Negative image off

28 Invisible image off

30 Black display (text)

31 Red display (text)

32 Green display (text)

33 Yellow display (text)

34 Blue display (text)

35 Magenta display (text)

36 Cyan display (text)

37 White display (text)

39 Default display (text)

40 Black background

41 Red background

42 Green background

43 Yellow background

44 Blue background

45 Magenta background

46 Cyan background

47 White background

49 Default background

Esc **[** *pi* **n**

(DSR) Device status report. Valid supported values for *pi* are:

5 Operating status. The *dtterm* utility responds with an *OK* message of ''*Esc* **[0 n**''.

6 (CPR) Cursor position report. The *dtterm* utility responds with the current cursor position in the form ''*Esc* **[** *p1* **;** *p2* **R**'' where *p1* is the current cursor line and *p2* is the current cursor row.

Esc **[** ? *pi* **n**

(DSR) DEC private device status report. Valid supported values for *pi* are:

15 Printer port status. The *dtterm* utility responds with a ''no printer available'' message of ''*Esc* **[** ? **13 n**''.

25 User-defined key status. The *dtterm* utility responds with either a message of ''*Esc* **[** ? **20 n**'' if UDKs are unlocked, or ''*Esc* **[** ? **21 n**'' if UDKs are locked.

26 Keyboard status. The *dtterm* utility responds with a message of ''*Esc* **[** ? **27 ; 1 n**'', which indicates a North American keyboard.

Esc **[** *p1* **;** *p2* **r**

(DECSTBM) Set top and bottom margins. The default value for *p1* is 1. The default value for *p2* is the current number of lines in the terminal window. The top and bottom margins are set to *p1* and *p2* respectively. Scrolling is not performed outside the margins.

Esc **[** *p1* **;** *p2* **;** *p3* **t**

Window manipulation. Valid values for *p1* (and any additional parameters) are:

1 Restore (de-iconify) window.

2 Minimise (iconify) window.

3 ; *x* **;** *y*
Move window to [*x*, *y*].

4 ; *height* **;** *width*
Resize the *dtterm* window to *height* and *width* in pixels.

5 Raise the *dtterm* window to the front of the stacking order.

6 Lower the *dtterm* window to the bottom of the stacking order.

7 Refresh the *dtterm* window.

8 ; *height* **;** *width*
Resize the text area to *height* and *width* in characters.

11 Report *dtterm* window state. If the *dtterm* window is open (non-iconified), it returns ''*Esc* **[1 t**''. If the *dtterm* window is iconified, it returns ''*Esc* **[2 t**''.

13 Report the *dtterm* window position. The terminal emulator returns ''*Esc* **[3 ;** *x* **;** *y* **t**''.

14 Report the *dtterm* window in pixels. The terminal emulator returns ''*Esc* **[4 ;** *height* **;** *width* **t**''.

18 Report the size of the area in characters. The terminal emulator returns ''*Esc* **[8 ;** *height* **;** *width* **t**''.

20 Report the *dtterm* window's icon label. The terminal emulator returns ''*Esc*] L *label Esc Backslash*''.

21 Report the *dtterm* window's title. The terminal emulator returns ''*Esc* **] l** *title Esc Backslash*''.

Esc **[** *pi* **x**

Request terminal modes. The default value is 0. Valid values are 0 or 1. If *pi* is 0, *dtterm* responds with the message of ''*Esc* **[2 ; 1 ; 1 ; 112 ; 112 ; 1 ; 0 x**''. If *pi* is 1, *dtterm* responds with the message of ''*Esc* **[3 ; 1 ; 1 ; 112 ; 112 ; 1 ; 0x**''.

Esc **[** ? *pi* **h**

(SM) DEC private set mode. This escape sequences sets DEC private modes. Valid supported values of *pi* are:

1 (DECCKM) Enable cursor keys mode. When cursor keys mode is enabled, the arrow keys send application sequences to the host.

3 (DECCOLM) Enable 132-column mode. When 132-column mode is enabled, the number of columns is the terminal window changed to 132. When entering into 132-column mode, the left, right, top, and

bottom margins are reset to their default positions and the display is cleared.

4 (DECSCLM) Enable smooth scrolling. When smooth scrolling is enabled, lines are added and the screen is scrolled a single line at a time.

5 (DECSCNM) Enable reverse video. When reverse video mode is enabled, the foreground and background colours of the terminal window are reversed.

6 (DECOM) Enable origin mode. When origin mode is enabled, the home cursor position is the upper-left corner of the screen, within the margins. The starting point for line numbers depends on the current top margin. The cursor cannot be moved outside the top and bottom margins.

7 (DECAWM) Enable autowrap. When autowrap mode is enabled, characters received when the cursor is at the right-most column of the page are inserted at the beginning of the next line. If the cursor is at the bottom line of the scrolling region, the page is scrolled up 1 line.

8 (DECARM) Enable auto-repeat keys. This option is ignored.

25 (DECTCEM) Enable cursor visible. In this mode, the text cursor is visible.

40 Enable DECCOLM escape sequence. When the DECCOLM escape sequence is enabled, the terminal emulator switches into either an 80- or 132-column window when it receives a DECCOLM escape sequence.

44 Enable margin bell. When the margin bell is enabled, the *dtterm* utility's bell (either audible or visible) is invoked when the cursor is a predefined distance from the right margin and a key is pressed.

45 Enable reverse-autowrap mode. When reverse-autowrap mode is enabled, and a backspace is received when the cursor is at the left-most column of the page, the cursor is wrapped to the right-most column of the previous line. If the cursor is at the top line of the scrolling region, the cursor is wrapped to the right-most column of the bottom line of the scrolling region. If the cursor is at the top line of terminal window, the cursor is wrapped to the right-most column of the bottom line of the terminal window.

46 Enable logging. When logging is enabled, all text received from the child process is logged to a file.

Esc **[** ? *pi* **l**

(RM) DEC private mode reset. This escape sequence sets DEC private modes. Valid supported values of *pi* are:

1 (DECCKM) Disable cursor keys mode. When cursor keys mode is disabled, the arrow keys send ANSI cursor sequences to the host.

3 (DECCOLM) Disable 132-column mode. When 132-column mode is disabled, the number of columns is the terminal window changed to 80. When entering into 80-column mode, the left, right, top, and

bottom margins are reset to their default positions and the display is cleared.

4 (DECSCLM) Disable smooth scrolling. When smooth scrolling is disabled, lines are added and the screen is scrolled up to a full screen at a time depending on how fast text is received from the child process.

5 (DECSCNM) Disable reverse video. When reverse video mode is disabled, the foreground and background colours of the terminal window are not reversed.

6 (DECOM) Disable origin mode. When origin mode is disabled, the home cursor position is the upper-left corner of the screen. The starting point for line numbers is independent of the current top margin. The cursor can be moved outside the top and bottom margins.

7 (DECAWM) Disable autowrap. When autowrap mode is enabled, characters received when the cursor is at the right-most column of the page, replace the character already on the line.

8 (DECARM) Disable auto-repeat keys. This option is ignored.

25 (DECTCEM) Disable cursor visible. In this mode, the text cursor is invisible.

40 Disable DECCOLM escape sequence. When the DECCOLM escape sequence is disabled, the terminal emulator ignores the DECCOLM escape sequence and does not switch into either an 80- or 132-column window when it is received.

44 Disable margin bell. When the margin bell is disabled, the *dtterm* utility's bell is not invoked when the cursor is a pre-defined distance from the right margin and a key is pressed.

45 Disable reverse-autowrap mode. When reverse-autowrap mode is disabled, and a backspace is received when the cursor is at the left-most column of the page, the cursor remains at that position.

46 Disable logging. When logging is disabled, text received from the child process is not logged to a file.

Esc **[?** *pi* **r**

Restore DEC private mode values. The value corresponding to mode *pi* previously saved is restored. Valid values for *pi* are the same as the DEC private modes supported by SM. Using this escape sequence is discouraged.

Esc **[?** *pi* **s**

Save DEC private mode values. The value corresponding to mode *pi* is saved. Valid values for *pi* are the same as the DEC private modes supported by SM. Using this escape sequence is discouraged.

Esc **]** *p1* **;** *p2* <*control*>-*G*

Set text parameters. This escape sequence allows various terminal emulator text values to be set. Valid supported values of *p1* are:

0 Change the icon name and window title to the string *p2*.

1 Change the icon name to the string *p2*.

2 Change the window title to the string *p2*.

3 Set the current working directory to the string *p2*. The terminal emulator tries to restart in this directory when it is restarted in a new session.

Esc ^ message Esc Backslash
(PM) Privacy message. The data received in a privacy message is ignored and is not displayed.

Esc _ pi Esc Backslash
(APC) Application program command. The terminal emulator implements no APC functions. The data is ignored and is not displayed.

Esc **[?** *pi* **K**
(DECSEL) Selective erase in line. The default value is 0. This escape sequence only erases erasable characters in a single line of text. Only those characters defined as erasable by the DECSCA escape sequence are erased. A parameter value of 0 erases from the cursor to the end of the line. A parameter value of 1 erases from the beginning of the line to the cursor position, inclusive. A parameter value of 2 erases the complete line.

Esc **[?** *pi* **J**
(DECSED) Selective erase in display. The default value is 0. This escape sequence only erases erasable characters in the display. Only those characters defined as erasable by the DECSCA escape sequence are erased. A parameter value of 0 erases from the cursor to the end of the display. A parameter value of 1 erases from the beginning of the display to the cursor position, inclusive. A parameter value of 2 erases the complete display.

Esc **] l** *text Esc Backslash*
Set the window title to *text*.

Esc **] I** *file Esc Backslash*
Set the icon to the icon found in file.

Esc **] L** *label Esc Backslash*
Set the icon name to *label*.

Esc **[! p**
(DECSTR) Soft terminal reset. This function performs a soft reset. For additional information, see Section 12.7.2.

12.7.2 Reset

The *dtterm* utility supports two levels of reset: full reset and soft reset. Reset can be invoked by menu buttons, the keyboard or by escape sequences. Soft reset performs the following actions:

- Turns on the text cursor (DECTCEM)
- Enables replace mode (IRM)
- Turns off origin mode (DECOM)
- Turns on autowrap (DECAWM)
- Turns off reverse wrap

- Unlocks the keyboard (KAM)
- Sets the cursor keypad mode to normal (DECCKM)
- Sets the numeric keypad mode to numeric (DECNKM)
- Sets the top and bottom margins to the first and last lines of the window (DECSTBM)
- Sets all character sets (GL, G0, G1, G2 and G3) to referenced ISO / IEC 646: 1983 standard
- Turns off all character attributes (SGR)
- Sets selective erase mode off (DECSCA)
- Clears any cursor state information saved with save cursor (DECSC)

Full reset performs the same functions as soft reset along with the following actions:

- Cursor is moved to the home position
- Clears the screen
- Clears user defined keys (DECUDK)
- Turns off reverse video (DECSCNM)
- Turns off auto linefeed mode (LNM)
- Turns on jump scroll (DECSCLM)

12.7.3 Transmitted Escape Sequences

12.7.3.1 Cursor Key Mode

The cursor keys transmit the following escape sequences depending on the setting of the mode specified, either via the **appCursorDefault** resource, or the mode specified via the DECCKM escape sequence.

Key	Normal	Application
Cursor Up	Esc **[A**	Esc **O A**
Cursor Down	Esc **[B**	Esc **O B**
Cursor Right	Esc **[C**	Esc **O C**
Cursor Left	Esc **[D**	Esc **O D**

12.7.3.2 Application Keypad Mode

The application keypad transmits the following escape sequences depending on the setting of the mode specified, either via the **appKeypadDefault** resource, or the mode specified via the DECPNM escape sequence.

Key	Numeric	Application
Space	Space	*Esc* **O A**
Tab	Tab	*Esc* **O I**
Enter	CR	*Esc* **O M**
PF1	*Esc* **O P**	*Esc* **O P**

PF2	*Esc* **O Q**	*Esc* **O Q**
PF3	*Esc* **O R**	*Esc* **O R**
PF4	*Esc* **O S**	*Esc* **O S**
* (multiply)	*	*Esc* **O j**
+ (add)	+	*Esc* **O k**
, (comma)	,	*Esc* **O l**
- (minus)	-	*Esc* **O m**
/ (divide)	/	*Esc* **O o**
0	0	*Esc* **O p**
1	1	*Esc* **O q**
2	2	*Esc* **O r**
3	3	*Esc* **O s**
4	4	*Esc* **O t**
5	5	*Esc* **O u**
6	6	*Esc* **O v**
7	7	*Esc* **O w**
8	8	*Esc* **O x**
9	9	*Esc* **O y**
= (equal)	=	*Esc* **O X**

12.7.3.3 Standard Function Keys

The function keys transmit the following escape sequences unless Sun function keys mode has been selected, either via the *dtterm* –**sk** option, or the **sunFunctionKeys** resource in *dtterm* or the DtTerm widget.

Key	Escape Sequence
F1	*Esc* **[1 1 ~**
F2	*Esc* **[1 2 ~**
F3	*Esc* **[1 3 ~**
F4	*Esc* **[1 4 ~**
F5	*Esc* **[1 5 ~**
F6	*Esc* **[1 7 ~**
F7	*Esc* **[1 8 ~**
F8	*Esc* **[1 9 ~**
F9	*Esc* **[2 0 ~**
F10	*Esc* **[2 1 ~**
F11	*Esc* **[2 3 ~**
F12	*Esc* **[2 4 ~**
F13	*Esc* **[2 5 ~**
F14	*Esc* **[2 6 ~**
F15	*Esc* **[2 8 ~**
F16	*Esc* **[2 9 ~**
F17	*Esc* **[3 1 ~**
F18	*Esc* **[3 2 ~**
F19	*Esc* **[3 3 ~**
F20	*Esc* **[3 4 ~**
Help	*Esc* **[2 8 ~**

Menu	*Esc* **[29~**
Find	*Esc* **[1~**
Insert	*Esc* **[2~**
Delete	*Esc* **[3~**
Remove	*Esc* **[3~**
Select	*Esc* **[4~**
Prior	*Esc* **[5~**
Next	*Esc* **[6~**

12.7.3.4 Sun Function Keys

Key	Escape Sequence
F1	*Esc* **[224z**
F2	*Esc* **[225z**
F3	*Esc* **[226z**
F4	*Esc* **[227z**
F5	*Esc* **[228z**
F6	*Esc* **[229z**
F7	*Esc* **[230z**
F8	*Esc* **[231z**
F9	*Esc* **[232z**
F10	*Esc* **[233z**
F11	*Esc* **[192z**
F12	*Esc* **[193z**
F13	*Esc* **[194z**
F14	*Esc* **[195z**
F15	*Esc* **[196z**
F16	*Esc* **[197z**
F17	*Esc* **[198z**
F18	*Esc* **[199z**
F19	*Esc* **[200z**
F20	*Esc* **[201z**
F21 (R1)	*Esc* **[208z**
F22 (R2)	*Esc* **[209z**
F23 (R3)	*Esc* **[210z**
F24 (R4)	*Esc* **[211z**
F25 (R5)	*Esc* **[212z**
F26 (R6)	*Esc* **[213z**
F27 (R7)	*Esc* **[214z**
F28 (R8)	*Esc* **[215z**
F29 (R9)	*Esc* **[216z**
F30 (R10)	*Esc* **[217z**
F31 (R11)	*Esc* **[218z**
F32 (R12)	*Esc* **[219z**
F33 (R13)	*Esc* **[220z**
F34 (R14)	*Esc* **[121z**
F35 (R15)	*Esc* **[122z**
Help	*Esc* **[196z**

Menu	*Esc*	**[1 9 7 z**
Find	*Esc*	**[1 z**
Insert	*Esc*	**[2 z**
Delete	*Esc*	**[3 z**
Remove	*Esc*	**[3 z**
Select	*Esc*	**[4 z**
Prior	*Esc*	**[5 z**
Next	*Esc*	**[6 z**

12.8 Capabilities

A conforming implementation of the XCDE terminal emulation services supports at least the following capabilities:

1. Provides users with a window that behaves like a terminal supporting the escape sequences specified in Section 12.7 on page 283, which are consistent with the the referenced ANSI X3.64-1979 standard and the referenced ISO / IEC 6429: 1992 standard.
2. Conforms to the Required items in the Application Style Checklist in Chapter 20 on page 355.
3. Has been internationalised using the standard interfaces in the X / Open CAE Specification, **System Interfaces and Headers**, **Issue 4, Version 2**, the X / Open CAE Specification, **Window Management: Xlib C Language Binding**, and the X / Open CAE Specification, **Motif Toolkit API**, and supports any locale available in the underlying X Window System Xlib implementation.
4. Allows users to customise the window in at least the following ways:
   a. Visibility of the cursor (visible or invisible)
   b. Shape of the cursor (box or underline)
   c. Blinking of the cursor (blinking on or off)
   d. Type of scrolling (smooth scrolling or jump scrolling)
   e. Type of bell (audible or visual)
   f. Enabling the margin bell (enable or disable)
   g. Position of the right margin for the margin bell
   h. Control sequences generated by the keyboard and numeric pad (normal or application)
   i. Locking the user-defined function keys (lock or unlock)
   j. End-of-line wrapping (enable or disable)
   k. Reverse end-of-line wrapping (enable or disable)
   l. Selection of end-of-line character to generate (carriage return or carriage return / linefeed)
   m. Size of the terminal window
   n. Normal or reverse-video operation
   o. Size of the font used for terminal operation
5. Supports the command-line options, resources and environment variables described in the man page *dtterm*.
6. Supports copy and paste operations for text.
7. Supports both soft and hard resets.
8. Supports XCDE font naming conventions for interface fonts, as described in Section 19.1 on page 343.

Chapter 13

Style Management Services

13.1 Introduction

The XCDE style management services allow users to customise the visual elements and system behaviour of the X/Open Common Desktop Environment. These services provide a graphical interface for customising the colours, fonts, backdrops and screen savers of the desktop and the system-wide behaviour of the keyboard and mouse. They also allow users to set the screen time-out interval and the session startup behaviour of XCDE and to customise the window behaviour on the desktop.

13.2 Actions

This section defines the actions that provide XCDE style management services to support application portability at the C-language source or shell script levels.

NAME

dtstyleaction — XCDE style management actions

SYNOPSIS

```
Dtstyle
```

DESCRIPTION

The XCDE Style Management Services support the following style management actions:

Dtstyle

Open a view of the desktop style manager.

These actions can be invoked from an application using the *DtActionInvoke*( ) function or invoked from a command line using the *dtaction* utility.

SEE ALSO

dtaction, *DtActionInvoke*( ) in the **XCDI** specification.

CHANGE HISTORY

First released in Issue 1.

13.3 Capabilities

A conforming implementation of the XCDE style management services supports at least the following capabilities:

1. Conforms to the Required items in the Application Style Checklist in Chapter 20 on page 355.
2. Has been internationalised using the standard interfaces in the X/Open CAE Specification, **System Interfaces and Headers**, **Issue 4, Version 2**, the X/Open CAE Specification, **Window Management: Xlib C Language Binding**, and the X/Open CAE Specification, **Motif Toolkit API**, and supports any locale available in the underlying X Window System Xlib implementation.
3. Allows the user to select a colour palette from a list of available colour palettes. Users can specify the number of colours used in the selected palette to be 8, 4 or 2 depending upon the variety of colours they prefer. Colour entries in the selected palette are used to colour the various components of the desktop. Desktop colours can be changed by selecting a different palette or modifying individual colours in the selected palette. Users can also create entirely new palettes. Selection of the colour palette and customisation of individual colours are immediately reflected on the desktop.
4. Allows the user to specify the desktop default font size. The XCDE style management services use the XCDE interface font aliases to provide visual integrity across the desktop. See Section 19.1.2 on page 348.
5. Allows the user to select a workspace's backdrop (root window) from a list of available backdrops.
6. Allows the user to customise the auto-repeat capability of the keyboard.
7. Allows the user to specify the mouse to be right-handed or left-handed and to specify the double-click interval, pointer acceleration, pointer movement threshold and meaning of button 2 (BTransfer or BSelect).
8. Allows the user to customise the beeper volume and duration.
9. Allows the user to enable or disable the screen saver, to select and preview from a list of available screen savers the screen saver to be run at screen time-out, and to set the screen saver time-out interval.
10. Allows the user to customise the window focus (focus follows mouse, click in window for focus), window behaviour (raise window with focus, allow primary windows on top, opaque move), and iconification (use icon box, place on workspace).
11. Allows the user to customise the session startup (resume current session, return to home session, prompt at logout time) and to enable/disable the logout confirmation dialog.
12. Allows users to replace their home session with the current session.

Chapter 14

Application Building Services

14.1 Introduction

The XCDE application building services provide an interactive, graphical environment that facilitates the development of XCDE-compliant applications. Two basic services are provided: aid in assembling graphical objects into the desired application user interface, and generation of appropriate calls to the routines that support XCDE desktop services (such as ToolTalk messaging, drag and drop, and so forth).

The key supported tasks for the application building services are:

- Interactive layout of the user interface for an application, constructing it piece-by-piece from a collection of objects from the XCDE and Motif toolkits.
- Managing an application *project* with its constituent *module* subdivisions accessible separately by multiple developers and able to be imported and exported to and from projects.
- Definition of window resizing behaviour.
- Definition of connections between objects to provide elements of application interface behaviour, and a limited test mode that allows connections to be exercised.
- Interactive specification of the interconnections desired between the application and CDE desktop services.
- Drag and drop specifications for individual objects.
- Editing of applications previously created using the XCDE application building services.
- Generation of C-language source code and associated project files (for example, makefiles and message catalogues) for the application.
- Generation (compilation) and invocation of the application from within the application building services, allowing the developer to execute the build/run/debug cycle all from a common environment (and without having to exit and restart the application building services).

14.2 Command-line Interfaces

This section defines the utility that provides XCDE application building services.

NAME

dtcodegen — generate code from an XCDE application building services project or module file

SYNOPSIS

```
dtcodegen [–changed] [–main] [–merge] [–nomerge] [–showall]
[–noshowall] [–s | –silent] [–v | –verbose] file ...

dtcodegen –help
```

DESCRIPTION

The *dtcodegen* utility reads files created by the XCDE application building services graphical user interface and produces C, Motif and XCDE source code for the user interface and application elements defined. The files supplied can be individual module files or a project file that contains references to zero or more module files.

OPTIONS

The *dtcodegen* utility does not support the X/Open Utility Syntax Guidelines because it uses the X Window System convention of full-word options. The following options are available:

–changed
Generate only source code for those modules that have changed since the last time *dtcodegen* was run.

–help
Write a help message to standard output explaining all *dtcodegen* options and then terminate.

–main
Produce the project files associated with the application's *main*( ) routine.

–merge
Merge generated stubs files with previous versions, perpetuating changes made or custom edits done to the previous stubs file. This is the default behaviour.

–nomerge
Do not merge existing and new **_stubs.c** files. This option overrides the default merging behaviour. If both **–merge** and **–nomerge** are used, the one given last on the command line takes precedence.

–showall
Cause the generated application to show (map) all application windows (main windows and dialogs) at startup, ignoring whether they are set to be initially visible or not. If no project is specified on the command line, *dtcodegen* performs as if **–showall** had been specified. (The **–noshowall** option suppresses this behaviour).

–noshowall
Cause the generated application to show at startup (map) only those windows (main windows and dialogs) whose initially visible attribute is true. If a project is specified on the command line, *dtcodegen* performs as if **–noshowall** had been specified. (The **–showall** option suppresses this behaviour).

–s | –silent
Work silently, producing no output except error messages while generating source code.

–v | –verbose
Be more verbose in providing progress and status messages during the generation of source code.

OPERANDS

The following operand is supported:

file A pathname of a project or module file.

RESOURCES

None.

STDIN

Not used.

INPUT FILES

All input files are text files in the format used by the XCDE application building services graphical user interface. See Section 14.4.1 on page 310.

ENVIRONMENT VARIABLES

The following environment variables affect the execution of *dtcodegen*:

LANG Provide a default value for the internationalisation variables that are unset or null. If *LANG* is unset or null, the corresponding value from the implementation-specific default locale will be used. If any of the internationalisation variables contains an invalid setting, the utility behaves as if none of the variables had been defined.

LC_ALL If set to a non-empty string value, override the values of all the other internationalisation variables.

LC_MESSAGES Determine the locale that is used to affect the format and contents of diagnostic messages written to standard error and informative messages written to standard output.

NLSPATH Determine the location of message catalogues for the processing of *LC_MESSAGES*.

ASYNCHRONOUS EVENTS

Default.

STDOUT

When –**help** is specified, *dtcodegen* writes to standard output a usage message in an unspecified format. Otherwise, standard output is not used.

STDERR

When –**verbose** is specified, *dtcodegen* writes to standard error informational progress messages and diagnostic messages in an unspecified format. Otherwise, standard error is used only for diagnostic messages.

OUTPUT FILES

The *dtcodegen* utility produces the following files:

*modname*_**ui.c**
The primary source code file for module *modname*, containing C code to create the objects in the module and establish connections for those objects.

*modname*_**ui.h**
Declarations and C externs for module *modname*.

*modname*_**stubs.c**
Callback functions for the element handlers specific to module *modname*.

*project***.c**
If *dtcodegen* is generating code for a project, this file contains *main*( ) plus any callback functions that are common across modules.

*project***.h**
If *dtcodegen* is generating code for a project, this file contains declarations for any callback functions and C externs that are common across interfaces.

.dtcodegen.log
A record of per-module code generation and the date and time of each module as it was processed. This data is required to provide support for the –**changed** option as part of determining which files need to be regenerated and which ones do not.

Additional application code should be added to the *modname***_stubs.c**, *project***.c** and *project***.c** files, as appropriate, because their contents are merged across runs of *dtcodegen*.

EXTENDED DESCRIPTION

None.

EXIT STATUS

The following exit values are returned:

0 successful completion

>0 an error occurred

CONSEQUENCES OF ERRORS

Because code generation involves the sequential production of a set of application files, errors that cause the *dtcodegen* utility to exit prematurely also may result in some module or project source files having been generated while others were not. Attempts to build the application from this mix of new and old generated code produce undefined results.

APPLICATION USAGE

Typically the *dtcodegen* utility is used indirectly through the XCDE application building services graphical user interface. This allows application code to be generated while the user is working with the Application Builder rather than through a separate interface or shell command line.

In some cases, however, it may be desirable to use the *dtcodegen* utility directly. A common example of this usage is to employ the code generator from within an application Makefile to produce a portion of the application code from pre-existing project or module files.

EXAMPLES

Run the code generator on the application defined by the project file **myproject.bip**:

```
dtcodegen myproject.bip
```

Run the code generator for the project in file **myproject.bip**, but only generate code for the module in file **modulename.bil**:

```
dtcodegen myproject.bip modulename.bil
```

Generate just the files associated with the main routine for the project in file **myproject-file**, namely **myproject.c** and **myproject.h**:

```
dtcodegen –main myproject-file
```

Search the current working directory for a project file and, if one is found, generate code for only those modules that have changed since the code generator was last run:

```
dtcodegen –changed
```

Generate, for the project in file **myproject.bip**, code only for those modules among the set of files named **module1**, **module2** and **module3** that have changed since the last time the code generator was run:

```
dtcodegen –changed myproject.bip module1 module2 module3
```

SEE ALSO

None.

CHANGE HISTORY

First released in Issue 1.

14.3 Actions

This section defines the actions that provide XCDE application building services to support application portability at the C-language source or shell script levels.

NAME

dtbuilderaction — XCDE application builder actions

SYNOPSIS

```
Dtbuilder [component]
Dtcodegen [component]
Open component
```

DESCRIPTION

The XCDE Application Builder Services support the following application builder actions:

Dtbuilder
Open an empty application builder view.

Dtbuilder *component*
Open an application builder view of the module or project named by the pathname in the *component* argument.

Dtcodegen
Prompt the user for the pathname of a project and generate code for the project specified by the user.

Dtcodegen *component*
Generate code for the project named by the pathname in the *component* argument.

Open *component*
Open an application builder view of the module or project named by the pathname in the *component* argument.

These actions can be invoked from an application using the *DtActionInvoke*( ) function or invoked from a command line using the *dtaction* utility.

SEE ALSO

dtaction, *DtActionInvoke*( ) in the **XCDI** specification.

CHANGE HISTORY

First released in Issue 1.

14.4 Capabilities

A conforming implementation of the XCDE application building services supports at least the following capabilities:

1. Provides application building services as described in the following subsections.
2. Conforms to the Required items in the Application Style Checklist in Chapter 20 on page 355, with the following exceptions that may exist on some implementations:
   a. There are certain operations that need not comply with checklist item 2-4, concerning the Input Device Model for keyboard-only users. The drag and drop of objects from the palette to the workspace and the ability to resize objects or move them around within a window or dialogue need not be available from the keyboard.
   b. Another possible exception to drag and drop style is described in Section 14.4.9 on page 322.
   c. The placement of secondary windows need not comply with checklist items ak and al, concerning Application Window Management.
3. Has been internationalised using the standard interfaces in the X/Open CAE Specification, **System Interfaces and Headers**, **Issue 4, Version 2**, the X/Open CAE Specification, **Window Management: Xlib C Language Binding**, and the X/Open CAE Specification, **Motif Toolkit API**, and supports any locale available in the underlying X Window System Xlib implementation.

14.4.1 Project and Module Files

The application building services support two file formats:

1. Application projects and modules are expressed in an implementation-dependent format. The application building services can read, write and drag and drop files in this format.
2. Modules can be imported from a file or exported to a file in the Motif User Interface Language (UIL) format. See the X/Open CAE Specification, **Motif Toolkit API**.

14.4.2 Project Management

The application building services provide the following project management capabilities:

1. manipulating a collection of modules as a single project that can be processed as a single entity
2. saving modules or the project, using existing or new names
3. removing modules from a project
4. Sharing a module among multiple projects
5. exporting a module from a project (Exporting is distinguished from saving because it typically uses a different file format, such as UIL. Exporting using the implementation-dependent format for modules is the equivalent of a regular save or save-as operation).
6. hiding modules from viewing (to simplify the workspace)
7. building (making, compiling, linking, and so forth) and executing the application.

14.4.3 Object Palette

The object palette is a panel that holds iconic representations for objects that can be instantiated as part of the application under construction. Construction is accomplished for these palette objects by dragging them from the palette and dropping them into the *workspace*, where the windows and dialogues of the application are arranged.

Objects on the palette include those in the following table.

Application Building Services Objects	
Object Type	**Motif Class**
Main Window	**XmMainWindow + ApplicationShell**
Dialogue Box	**XmDialogShell**
File Selection Dialogue	**XmFileSelectionBox + XmDialogShell**
Control Pane	**XmForm**
Text Pane	**XmText**
Drawing Pane	**XmDrawingArea**
Term Pane	**DtTerm**
Button	1. **XmPushButton** 2. **XmDrawnButton** 3. **XmArrowButton**
Menu Button	**DtMenuButton**
Combo Box	**DtComboBox**
Option Menu	**XmOptionMenu**
Radio Box	**XmRowColumn + XmToggleButton(s)**
Check Button	**XmRowColumn + XmToggleButton(s)**
Gauge	**XmScale**
Scale	**XmScale**
Separator	**XmSeparator**
Text Field	**XmTextField**
Label	**XmLabel**
List Box	**XmList + XmScrolledWindow**
Spin Box	**DtSpinButton**

The column labelled Motif Class describes the functionality of the object in terms of named Motif widgets. The object created includes the combined user interface behaviour of the named widgets, although the implementation may include additional or alternative widgets to achieve this behaviour. An object in the table with several numbered widgets listed as its Motif Class means that implementations may select the type of widget used to support the object, based on the layout behaviour and properties selected by the developer. An object with a ‘‘Dt’’ widget listed in its Motif Class need not be supported in exported modules using the Motif UIL format.

There are other objects created for the application that need not be available directly on the object palette. These include:

- groups of objects
- menus
- message boxes
- paned windows.

Unless noted otherwise, all references to ‘‘objects’’ in this Capability section apply to the palette objects as well as these additional objects.

14.4.4 Object Layout

The developer can position palette objects placed on the workspace as follows:

1. The drag and drop operation establishes the initial position.
2. The developer can center objects and align them with each other, horizontally or vertically.
3. The developer can specify relative positioning of objects.
4. The developer can group objects and affect their positions relative to each other following window resize operations performed by the developer or the application end-user.

14.4.5 Object Properties

The XCDE application building services allow the developer to display and modify a wide range of attributes for each object, some of which correspond to X Windows resources for the underlying Motif or XCDE widgets. For each of the following objects, the developer can specify the associated attributes:

Main Window

1. Object name
2. Window title
3. Icon filename
4. Icon label
5. User resize mode (fixed/adjustable)
6. Window areas present (menubar, toolbar, footer)
7. Geometry: x, y, width, height
8. Initial state (iconic, visible)
9. Background colour
10. Foreground colour
11. Help text
12. Additional help information: help volume, location ID

Custom Dialogue

1. Object name
2. Dialogue title
3. Window parent
4. User resize mode (fixed/adjustable)
5. Dialogue areas present (button panel, footer)
6. Default button (choose among the buttons in the dialogue)
7. Geometry: x, y, width, height
8. Initial state (visible)
9. Background colour

10. Foreground colour
11. Help text
12. Additional help information: help volume, location ID

File Chooser

1. Object name
2. Window parent
3. Title
4. Initial directory
5. Search pattern type (files, folders, both)
6. Search pattern
7. OK button label
8. Initial state (visible)
9. Popdown behaviour (automatically dismiss)
10. Background colour
11. Foreground colour
12. Help text
13. Additional help information: help volume, location ID

Control Pane

1. Object name
2. Geometry (x,y,width height)
3. Initial state (visible, active)
4. Border frame (none/shadow out/shadow in/etched out/etched in)
5. Background colour
6. Name of attached menu
7. Help text
8. Additional help information: help volume, location ID

Text Pane

1. Object name
2. Scrollbars (always/never)
3. Position: x, y
4. Size: width and height in either characters or pixels
5. Word wrap
6. Operation (read-write/read-only)
7. Initial value

8. Initial state (visible, active)
9. Border frame (none/shadow out/shadow in/etched out/etched in)
10. Background colour
11. Foreground colour
12. Name of attached menu
13. Help text
14. Additional help information: help volume, location ID

Draw Area Pane

1. Object name
2. Scrollbars (always/as needed/never)
3. Position: x, y
4. Visible size: width, height
5. Total size: width, height
6. Initial state (visible, active)
7. Border frame (none/shadow out/shadow in/etched out/etched in)
8. Background colour
9. Foreground colour
10. Name of attached menu
11. Help text
12. Additional help information: help volume, location ID

Label

1. Object name
2. Size policy (size of label/fixed)
3. Geometry: x, y, width, height
4. Label type (string/graphic)
5. Label
6. Label alignment (left/right/centered)
7. Initial state (visible, active)
8. Background colour
9. Foreground colour
10. Help text
11. Additional help information: help volume, location ID

Spin Box

1. Object name

2. Label type (string/graphic)
3. Label Position (left/above)
4. Spin box type (numeric/string list)
5. Arrow Style (flat beginning/flat end/beginning/end/split)
6. Value range: minimum, maximum
7. Increment value: integer portion and decimal point portion
8. Initial value
9. Items list
10. Geometry: x, y, width, height
11. Initial state (visible, active)
12. Background colour
13. Foreground colour
14. Help text
15. Additional help information: help volume, location ID

Choice

1. Object name
2. Choice type (radio box/check box/option menu)
3. Position: x, y
4. Label type (string/graphic)
5. Label
6. Label position (left/above)
7. Layout (rows/columns)
8. Number of rows/columns
9. Initial state (visible, active)
10. Background colour
11. Foreground colour
12. List of choice items, specifying for each:
    a. Item label (string/graphic)
    b. Item state (active, selected)
    c. Item help text
    d. Additional item help information: help volume, location ID
13. Help text
14. Additional help information: help volume, location ID

Separator

1. Object name
2. Geometry: x, y, width, height
3. Line style (none/etched in/etched out/etched in dash/etched out dash/single line/double line/single dashed line/double dashed line)
4. Orientation (horizontal/vertical)
5. Initial state (visible, active)
6. Background colour
7. Foreground colour
8. Help text
9. Additional help information: help volume, location ID

Button

1. Object name
2. Button type (push/drawn/menu)
3. Size policy (size of label/fixed)
4. Geometry: x, y, width, height
5. Label type (string/graphic/arrow)
6. Label
7. Arrow direction (up/down/left/right)
8. Label alignment (left/right/centered)
9. Initial state (visible, active)
10. Background colour
11. Foreground colour
12. Help text
13. Additional help information: help volume, location ID

Combo Box

1. Object name
2. Label type (string/graphic)
3. Label
4. Label position (left/above)
5. Combo box type (static/editable)
6. List items, choosing for each
    a. Item label
    b. Item state (selected/not selected)
7. Geometry: x, y, width, height

8. Initial state (visible, active)
9. Background colour
10. Foreground colour
11. Help text
12. Additional help information: help volume, location ID

Text Field

1. Object name
2. Position: x, y
3. Width
4. Width units (characters / pixels)
5. Label type (string / graphic)
6. Label
7. Label position (left / above)
8. Operation (editable / read-only)
9. Maximum characters
10. Initial value
11. Initial state (visible, active)
12. Background colour
13. Foreground colour
14. Attached menu name
15. Help text
16. Additional help information: help volume, location ID

Scale

1. Object name
2. Label type (string / graphic)
3. Label position (left / above)
4. Scale type (scale / gauge)
5. Orientation (horizontal / vertical
6. Direction (left to right / right to left / top to bottom / bottom to top)
7. Value range: minimum, maximum
8. Increment value: integer portion, decimal point portion
9. Initial value
10. Show value (yes / no)
11. Geometry: x, y, width, height

12. Initial state (visible, active)
13. Background colour
14. Foreground colour
15. Help text
16. Additional help information: help volume, location ID

List

1. Object name
2. Selection mode (single select / browse select / multiple select / browse multiple select)
3. Width policy (longest item / fixed)
4. Geometry: x, y, width, height
5. Label type (string / graphic)
6. Label
7. Label position (left / above)
8. Number of items visible
9. Initial state (visible, active)
10. Background colour
11. Foreground colour
12. Attached menu name
13. List of items, specifying for each
    a. Item label
    b. Item state (selected / not selected)
14. Help text
15. Additional help information: help volume, location ID

Menu

1. Object name
2. Tearoff (enabled / disabled)
3. Background colour
4. Foreground colour
5. List of items, specifying for each
    a. Item label
    b. Item mnemonic
    c. Item accelerator
    d. Item submenu
    e. Item initial state (active)

6. Help text
7. Additional help information: help volume, location ID

Group

1. Object name
2. Position: x, y
3. Layout type (as is / vertical / horizontal / grid)
4. Vertical alignment (left / on colons / center / right)
5. Horizontal alignment (top / center / bottom)
6. Grid arrangement (rows / columns)
7. Number of rows / columns in grid
8. Spacing type (proportional / absolute)
9. Spacing value
10. Border frame (none / shadow out / shadow in / etched out / etched in)
11. Initial state (visible, active)
12. Help text
13. Additional help information: help volume, location ID

Paned Window

1. Object name
2. List of panes
3. Pane geometry: x, y
4. Pane height: minimum, maximum
5. Help text
6. Additional help information: help volume, location ID

Message Dialogues

1. Object Name
2. Dialogue title
3. Message type (error / information / working / question / warning)
4. Message text
5. Action #1 enabled (yes / no)
6. Action #1 name
7. Action #2 enabled (yes / no)
8. Action #2 name
9. Cancel action enabled (yes / no)
10. Help action enabled (yes / no)

11. Default button (choosing among all those present and enabled)
12. Help text
13. Additional help information: help volume, location ID

14.4.6 Browser Window

The browser window provides a tree-structured representation of the objects in a project, using glyphs to identify each type of object, and supports editing and manipulation of objects.

More than one browser can be open at a time, but a browser need not support the display of more than one module at a time. If a module is being viewed by multiple browsers on the desktop, updates in one browser cause the appropriate changes in the other browsers.

The browser window supports the following operations:

1. View the objects by instance name or object type only
2. Expand/collapse subtrees in the hierarchy to hide them from view
3. Change the name of an object
4. Edit the properties of an object
5. Delete an object from a module

14.4.7 Application Framework

The XCDE application building services create one or more C-language source files that the developer uses as a framework for the application; the framework includes sufficient source code to implement the visual appearance and interactive characteristics of the application, but lacks the specific data processing logic required to achieve all of the application's business purpose. The developer can cause the XCDE application building services to include additional framework source code that addresses the following XCDE-related activities:

ToolTalk
: The developer can specify one of the following desktop message handling capabilities desired for the application:

  - No message handling
  - *Basic* participation in the ToolTalk desktop message set; messages are handled in the default manner
  - *Advanced* participation in the ToolTalk desktop message set; a connection is established, but the application is expected to handle its messages in a custom manner

Session Management
: The developer can control application participation in desktop session management, choosing among no sessioning, command-line based sessioning (ICCCM), save-file based sessioning (CDE) and both command-line and save-file sessioning. If sessioning is desired, the developer can identify the functions to be used to save and restore application state.

Generated Code Resource File Control

The developer can control which object-related resources (in the X and Motif sense) will be written out to an application resource file. Object resources are identified by type (colours, label strings, initial values, geometry, other strings, and other) and each type can be selected to appear in the application resource file produced by the code generator.

Internationalisation

The developer can cause the code generator to determine whether internationalisation message handling functions should be used in the application and whether a message source text file should be generated.

Drag and Drop

The developer can specify for any application user interface object:

1. The operations the object, acting as a drag source, is prepared to support: Move, Copy, Link.
2. The pathname of the pixmap to be used as the base of the composited drag icon.
3. The data types into which the application is prepared to convert the drag object at the request of the receiving application.
4. The operations the object, acting as a drop destination, is prepared to support: Move, Copy, Link.
5. The data types the receiving application is prepared to receive.
6. An indication that dropping onto a child object should be treated as a drop onto the parent object, provided the child does not have its own drop properties explicitly defined.

Help

The developer can associate help text with any object. At least three types of help can be established:

1. Context-sensitive help — Help information for a specific user interface element, provided for the object with input focus whenever the user presses the <help> key.
2. Application help — Help information above and beyond context-sensitive help and provided by the application either when the user presses a button labeled ‘‘Help’’ (typically on a dialogue) or selects a help-related item from a menu.
3. On-item help — An interactive mode in which the application allows the user to select (with the mouse) an object for which help is needed. This allows the user to get context-sensitive help on an object that cannot take input focus (for example, a control pane or gauge).

Messages

The developer can associate message text with any object. The following types of messages can be constructed.

Question

A question that can be answered with a simple yes or no.

Warning

A request for confirmation before dangerous actions are performed.

Information
: A message where the user's response will not alter application actions.

Error
: A notification that a requested action could not be performed.

Working
: An indication of the progress of a requested action.

Each of the preceding five message types is populated with buttons appropriate for the category: Help, Cancel and generic action buttons. The developer can associate a callback routine name with each button.

14.4.8 Connections

The XCDE application building services allow the developer to specify interrelationships that should exist between two objects. Each connection is defined conceptually as a rule of the form:

```
For <SourceObject> when <Event> perform <Action> on <TargetObject>
```

where:

SourceObject
: The application or an object defined in the application

Event
: Any of a set of occurrences, typically consistent with the Xt Intrinsics and Motif event model, and tailored to suit the type of the *SourceObject*

Action
: Any of a set of instructions that should be carried out, consistent with the Xt Intrinsics and Motif event mode, but augmented to include ''pseudo-events'' and suit the type of the *TargetObject*

TargetObject
: The object upon which the action should be taken

An example of such a rule is:

```
For HelpPushButton when ClickSelect perform PopUp on HelpDialog
```

14.4.9 Drag and Drop Capabilities

The XCDE application building services provide standard XCDE drag and drop capabilities as follows:

1. The developer can drag project files to the application building services, which in turn loads a new project. The application building services need not support multiple projects open simultaneously.
2. The developer can drag module files to the main window of the application building services, which in turn includes the module in the current project. (This is equivalent to importing a module.)

When the developer drags objects from the palette to the workspace, this action does not use the standard XCDE drag and drop facilities; because of the requirements for exact positioning of the dropped object, there are implementation-dependent differences in the visual feedback given to the developer as the drag proceeds.

Chapter 15

Application Integration Services

15.1 Introduction

The XCDE application integration services allow application developers and system administrators to integrate applications into the X/Open Common Desktop Environment. These services provide a utility for making an application's actions and datatypes, icons and help volumes available for use in the desktop.

15.2 Command-line Interfaces

This section defines the utility that provides XCDE application integration services.

NAME

dtappintegrate — integrate applications into the XCDE

SYNOPSIS

`dtappintegrate –s` *`application_root`* **[**`–t` *`target_path`***]** **[**`–l` *`locale`***]** **[**`–u`**]** **[**`–?`**]**

DESCRIPTION

The *dtappintegrate* utility integrates applications into XCDE. Application installation scripts should invoke *dtappintegrate* as the last step before exiting. The *dtappintegrate* utility requires appropriate privileges.

When *dtappintegrate* is invoked with no *target_path* specified, it creates symbolic links to the application's XCDE configuration files under the following default XCDE system locations:

/etc/dt/appconfig/types/<*locale*>
Contains symbolic links to the application action and datatype files

/etc/dt/appconfig/appmanager/<*locale*>
Contains symbolic links to the application group subdirectory

/etc/dt/appconfig/help/<*locale*>
Contains symbolic links to the application help files

/etc/dt/appconfig/icons/<*locale*>
Contains symbolic links to the application icons

OPTIONS

The *dtappintegrate* utility supports the X/Open Utility Syntax Guidelines. The following options are available:

–s *application_root*
Integrate the application files that are located under *application_root*. The *application_root* is the top directory under which all of an application's files are installed. The *dtappintegrate* utility looks for application XCDE configuration files in the following subdirectories, with all C locale subdirectories containing the application's default XCDE configuration files:

<*application_root*>**/dt/appconfig/types/**<*locale*>
Contains application action and datatype files

<*application_root*>**/dt/appconfig/appmanager/**<*locale*>
Contains application group files

<*application_root*>**/dt/appconfig/icons/**<*locale*>
Contains application icons

<*application_root*>**/dt/appconfig/help/**<*locale*>
Contains application help files

–t *target_path*
Link the application XCDE configuration files to *target_path* rather than to the default XCDE system locations.

–l *locale*
Integrate only the files found in the *locale* subdirectories. If this option is not specified, all of the application's XCDE configuration files are integrated.

–u Destroy the symbolic links previously created by *dtappintegrate*. If **–l** is specified with the **–u** option, only the symbolic links to the XCDE configuration files in the specified *locale* subdirectories are destroyed.

–? Write a help message to standard output that describes the command syntax of *dtappintegrate* and exit.

OPERANDS

None.

STDIN

Not used.

INPUT FILES

None.

ENVIRONMENT VARIABLES

None.

RESOURCES

None.

ASYNCHRONOUS EVENTS

Default.

STDOUT

When no option or the **–?** option is used, *dtappintegrate* writes to standard output a usage message.

During execution, *dtappintegrate* writes confirmation messages to standard output.

STDERR

Used only for diagnostic messages.

OUTPUT FILES

The *dtappintegrate* utility creates the symbolic links to the application's XCDE configuration files.

EXTENDED DESCRIPTION

None.

EXIT STATUS

The following exit values are returned:

0 Successful completion.

2 Help message displayed.

3 Not invoked with appropriate privileges.

4 Invalid option.

CONSEQUENCE OF ERRORS

Default.

APPLICATION USAGE

None.

EXAMPLES

None.

SEE ALSO

None.

CHANGE HISTORY

First released in Issue 1.

15.3 Actions

This section defines the actions that provide XCDE application integration services to support application portability at the C-language source or shell script levels.

NAME

dtappaction — XCDE application management actions

SYNOPSIS

```
Dtappmgr
ReloadApps
```

DESCRIPTION

The XCDE Application Integration Services support the following application management actions:

Dtappmgr

Open a view of the Application Manager.

ReloadApps

Reload the database of action and data types, update the Application Manager and update the index of help information.

These actions can be invoked from an application using the *DtActionInvoke*( ) function or invoked from a command line using the *dtaction* utility.

SEE ALSO

dtaction, *DtActionInvoke*( ) in the **XCDI** specification.

CHANGE HISTORY

First released in Issue 1.

Chapter 16

Action Creation Services

16.1 Introduction

The XCDE action creation services allow users to create and modify actions and datatypes that are used in the X/Open Common Desktop Environment. Actions provide the ability to associate an application with an icon on the desktop. Datatypes provide the ability to associate a data file possessing specific attributes with an icon on the desktop.

This service has a graphical interface for gathering information about the actions and datatypes the user is defining. The interface allows the user to define an application action and the datatypes for the data files associated with that application.

Users can choose to define actions and datatypes for personal or system-wide use. Actions and datatypes created for system-wide use should be installed using the *dtappintegrate* utility (see Chapter 15 on page 323).

16.2 Actions

This section defines the actions that provide XCDE action creation services to support application portability at the C-language source or shell script levels.

NAME

dtactionaction — XCDE action management actions

SYNOPSIS

```
Dtcreate [file]
ReloadActions
```

DESCRIPTION

The XCDE Action Creation Services support the following action management actions:

Dtcreate

Open an empty view of the desktop action and data type editor.

Dtcreate *file*

Open a view of the desktop action and data type editor and load the action and data type description file named by the pathname in the *file* argument. The description file must have been previously created by the desktop action and data type editor.

ReloadActions

Reload the database of desktop actions.

These actions can be invoked from an application using the *DtActionInvoke*( ) function or invoked from a command line using the *dtaction* utility.

SEE ALSO

dtaction, *DtActionInvoke*( ) in the **XCDI** specification.

CHANGE HISTORY

First released in Issue 1.

16.3 Capabilities

A conforming implementation of the XCDE action creation services supports at least the following capabilities:

1. Conforms to the Required items in the Application Style Checklist in Chapter 20 on page 355.
2. Has been internationalised using the standard interfaces in the X/Open CAE Specification, **System Interfaces and Headers**, **Issue 4, Version 2**, the X/Open CAE Specification, **Window Management: Xlib C Language Binding**, and the X/Open CAE Specification, **Motif Toolkit API**, and supports any locale available in the underlying X Window System Xlib implementation.
3. Creates action and datatype definitions files.
4. Allows modification of action and datatype definition files that were previously created using the XCDE action creation services.
5. Allows specification in an action definition of the application, command, shell script, etc., to be executed when the action is opened.
6. Allows specification in an action definition of the window type (graphical, terminal emulator or no output) to be used for displaying output when the action is opened.
7. Allows specification in an action definition of the datatypes, such as an application's data files, that use the action.
8. Allows specification in an action definition of the icon to be used for representing the action.
9. Allows specification in an action definition of the icon label to be used for representing the action.
10. Allows specification in an action definition of help information to be associated with the action icon.
11. Allows specification in a datatype definition of the datatype characteristics based on name pattern, file permission pattern and/or file contents.
12. Allows specification in a datatype definition of the command to be used to print the datatype.
13. Allows specification in a datatype definition of the icon to be used for representing the datatype.
14. Allows specification in a datatype definition of help information to be associated with the datatype icon.
15. Allows for invocation of the **ReloadApps** action, so that newly created or modified actions and datatypes become immediately available.

Chapter 17

Print Job Services

17.1 Introduction

The XCDE print job services provide information to users about printers and print jobs. These services have a graphical interface for viewing the properties of printers and print jobs, and for deleting print jobs.

17.2 Actions

This section defines the actions that provide XCDE print job services to support application portability at the C-language source or shell script levels.

NAME

dtprintinfoaction — XCDE print job actions

SYNOPSIS

```
Dtprintinfo [printer]
DtPrintManager
Print file
```

DESCRIPTION

The XCDE Print Job Services support the following action for viewing printers and print jobs:

Dtprintinfo
Display the default printer and its print jobs.

Dtprintinfo *printer*
Display the printer named by the printer in the *printer* argument and its print jobs.

DtPrintManager
Display all configured printers.

Print
Display the default printer and its print jobs.

Print *file*
Submit the file named by the pathname in the *file* argument to the default printer.

These actions can be invoked from an application using the *DtActionInvoke*( ) function or invoked from a command line using the *dtaction* utility.

SEE ALSO

dtaction, *DtActionInvoke*( ) in the **XCDI** specification.

CHANGE HISTORY

First released in Issue 1.

17.3 Capabilities

A conforming implementation of the XCDE print job services supports at least the following capabilities:

1. Conforms to the Required items in the Application Style Checklist in Chapter 20 on page 355.
2. Has been internationalised using the standard interfaces in the X/Open CAE Specification, **System Interfaces and Headers**, **Issue 4, Version 2**, the X/Open CAE Specification, **Window Management: Xlib C Language Binding**, and the X/Open CAE Specification, **Motif Toolkit API**, and supports any locale available in the underlying X Window System Xlib implementation.
3. Displays the list of configured printers. The user can select which printers will be displayed.
4. Allows viewing of the list of print jobs for each printer being displayed. Users can view all jobs or only their own jobs.
5. Allows deletion of displayed print jobs, based on appropriate permissions.
6. Allows the user to display selectively only one printer upon invocation of the XCDE print job services.

Chapter 18

Calculator Services

18.1 Introduction

The XCDE calculator services provide basic computation capabilities to users of the X/Open Common Desktop Environment desktop. They are designed to address the needs of professionals in business, engineering and computer science.

18.2 Actions

This section defines the actions that provide XCDE calculator services to support application portability at the C-language source or shell script levels.

NAME

dtcalcaction — XCDE calculator actions

SYNOPSIS

```
Dtcalc
```

DESCRIPTION

The XCDE Action Creation Services support the following calculator actions:

Dtcalc

Open a view of the desktop calculator tool.

These actions can be invoked from an application using the *DtActionInvoke*( ) function or invoked from a command line using the *dtaction* utility.

SEE ALSO

dtaction, *DtActionInvoke*( ) in the **XCDI** specification.

CHANGE HISTORY

First released in Issue 1.

18.3 Capabilities

A conforming implementation of the XCDE calculator services supports at least the following capabilities:

1. Provides calculator services as described in the following subsections.
2. Conforms to the Required items in the Application Style Checklist in Chapter 20 on page 355.
3. Has been internationalised using the standard interfaces in the X/Open CAE Specification, **System Interfaces and Headers**, **Issue 4, Version 2**, the X/Open CAE Specification, **Window Management: Xlib C Language Binding**, and the X/Open CAE Specification, **Motif Toolkit API**, and supports any locale available in the underlying X Window System Xlib implementation.

18.3.1 General Calculator Capabilities

The following general capabilities are supported by the calculator:

1. The user can perform calculations using a keypad and display area modelled after a desk calculator.
2. The user can input from the keyboard as well as through buttons in the calculator window.
3. The user can input numbers in binary, octal, decimal and hexadecimal bases. Decimal is the default.
4. Calculations are executed in the order that they are entered. The user can use parentheses to control the order of operations.
5. The user can set the accuracy of calculations.
6. Arithmetic, scientific, logical and financial functions, described in the following subsections, may be grouped into modes. The arithmetic functions are always available.
7. The user can access at least ten memory registers.
8. The user can store, recall and exchange values between a specified register and the display area.
9. The user can view the contents of all memory registers.

18.3.2 Arithmetic Operations

The user can perform the following arithmetic operations:

1. Addition
2. Subtraction
3. Multiplication
4. Division
5. Reciprocal
6. Square root
7. Square
8. The value of a specified percentage of a number (for example, 32 percent of 50)

9. Change the sign of a number
10. Integer component of a number
11. Fractional component of a number
12. Absolute value

18.3.3 Scientific Operations

The user can perform the following scientific calculation operations:

1. e^x
2. 10^x
3. y^x
4. Factorial
5. Trigonometric functions:
    a. Sine, cosine and tangent
    b. Arc sine, arc cosine and arc tangent
    c. Hyperbolic sine, hyperbolic cosine and hyperbolic tangent
6. Natural log and log base 10
7. Random number between 0 and 1

The user can select one of three trigonometric bases when in scientific mode: degrees, radiants or gradients. Degrees is the default.

18.3.4 Financial Operations

The user can perform the following financial calculation operations:

1. Time value of money based on
    a. Number of periods
    b. Annual interest rate
    c. Present value
    d. Payment amount
    e. Future value
    f. Payments per year
2. Depreciation using the following rules:
    a. Double declining balance
    b. Straight line
    c. Sum of years digits

The user can view and clear all financial registers.

18.3.5 Logical Operations

The user can perform the following logical operations:

1. Bitwise logical OR, AND, NOT, XOR and XNOR
2. Bitwise shift left and right
3. Truncate value to 16 or 32 bits

Chapter 19

Application Conventions

This chapter lists font and icon conventions for XCDE applications.

19.1 Font Conventions

19.1.1 Standard Application Font Names

The XCDE Standard Application Font Names are a set of generic X Window System font names, usable by applications as their default fonts, for the most common categories of type designs and styles. These names, for at least six sizes of 13 typefaces, must be provided on all XCDE systems. They are typically mapped to existing fonts on the system using the font alias mechanism, although this method is not required.

The XCDE Standard Application Font Names described here allow applications to use a single set of default font specifications in their *app-defaults* files, without concern for the system on which XCDE is running. These *app-defaults* application defaults are given as XLFD font name patterns that will match the standard XCDE font names on all XCDE systems.

19.1.1.1 Background

Application fonts are the fonts used within an application, where a wide variety of text designs, styles, weights and point sizes are useful. These variations are used for emphasis, cross-references, section headers, and so forth. The standard names attempt to provide the minimum variety in generic designs, styles and sizes that an application might want to use as defaults. (The XCDE Standard Interface Font Names, described in Section 19.1.2 on page 348, provide a similar mechanism for the elements of the XCDE desktop itself.)

19.1.1.2 Rationale

Two of the most common design variations in fonts used to display text are the presence or absence of serifs and the choice between proportional or regularly spaced (mono-spaced) characters. Combining these two design variations yields four ‘‘generic’’ font designs, or families:

- serif proportionally-spaced
- sans serif proportionally-spaced
- serif mono-spaced
- sans serif mono-spaced

Common examples of these four designs are:

- Times Roman
- Helvetica
- Courier
- Lucida Sans Typewriter

Each of these designs typically come, for text fonts, in four styles (combinations of weight and slant):

- plain
- bold
- italic
- bold-italic

The four styles of each of the four design variations yield 16 generic font variations. These 16 generic fonts are among the most commonly used in general desktop computing.

In some cases, applications do not care about the exact font family or name to be used, but do need to use a mono-spaced font, a sans serif font or a serif font. This XCDE mechanism allows such applications to be freed from the need to be concerned about the exact font names that may or may not be present on a particular XCDE system.

19.1.1.3 The Standard Names for the Latin-1 Character Set

The 13 standard application font names are provided on all XCDE systems only for the referenced ISO/IEC 8859-1:1987 standard (Latin-1) character set. These represent 12 generic design and style variations (serif and sans serif proportionally-spaced, and a mono-spaced font that is either serif or sans serif), as well as a symbol font. These standard names are provided in addition to the ''real'' names of the fonts that the standard names are mapped to for a particular XCDE system.

19.1.1.4 XLFD Field Values for the Standard Application Font Names

The standard names are available using the X Window System XLFD font naming scheme. There are three aspects to the standard names:

- The *underlying font* on each system, to which a standard name is mapped, typically will be different on each system.
- The *standard name* itself, a full XLFD name mapped to the underlying font, may be different on each system in some of the XLFD fields. However, most of the fields are the same from system to system, allowing the *patterns* (described next) to be the same.
- The font resource *pattern* containing the * wildcards, used in *app-defaults* files, which will match the full XLFD name of the standard name, is the same across all systems, for a given use in an *app-defaults* file.

Systems must provide full XLFD names for the standard names, mapped to system-dependent underlying fonts, so that the XLFD patterns used in XCDE application *app-defaults* files will always match one of the full XLFD names provided.

The Standard Application Font Names are identified by the presence of the following XLFD field name values:

- FOUNDRY is **dt**
- FAMILY_NAME is **application**
- WEIGHT_NAME is **medium** or **bold**
- SLANT is **r** or **i**
- SETWIDTH is **normal**

- ADD_STYLE is **sans** for sans serif, **serif** for serif
- SPACING is **p** or **m**
- CHARSET_REGISTRY is **iso8859**
- CHARSET_ENCODING is **1**

Although **sans** and **serif** are not required by the XLFD font convention, they are always part of the standard XCDE font names.

19.1.1.5 Point Sizes

The complete set of point sizes available for each of the standard application font names is determined by the set of fonts included in a system, whether bitmapped only or both bitmapped and scalable outline. The minimum set of sizes required and available on all XCDE systems corresponds to the standard sizes of bitmapped fonts that make up the default mapping for X11R5: 8, 10, 12, 14, 18 and 24.

For example, the entire set of six sizes of the plain monospaced font, on any XCDE system, is represented by:

```
-dt-application-medium-r-normal-*-*-80-*-*-m-*-iso8859-1
-dt-application-medium-r-normal-*-*-100-*-*-m-*-iso8859-1
-dt-application-medium-r-normal-*-*-120-*-*-m-*-iso8859-1
-dt-application-medium-r-normal-*-*-140-*-*-m-*-iso8859-1
-dt-application-medium-r-normal-*-*-180-*-*-m-*-iso8859-1
-dt-application-medium-r-normal-*-*-240-*-*-m-*-iso8859-1
```

These patterns will match the corresponding standard font name on any XCDE system, even though the PIXEL_SIZE and AVERAGE_WIDTH numeric fields may be different on various systems, and the matched fonts may be either serif or sans serif, depending on the implementation of the set of standard names. The RESOLUTION fields in the XLFD names of the underlying fonts, when those fonts are bitmapped fonts, must match the resolution of the monitor on which the fonts are displayed for the point sizes to be accurate. To provide expected point size behaviour for applications, systems should ensure that the RESOLUTION_X and RESOLUTION_Y fields of the underlying fonts vary no more than 20% from the real monitor resolution of the displays on which the fonts will be used.

Applications requesting point sizes different from the six in the minimum set may obtain either ‘‘scaled bitmapped’’ fonts of the requested design, or scaled outline versions of the requested design. This behaviour requires that the X server in question support the scaling of fonts and that the standard names are mapped to underlying fonts that can be scaled using this support.

19.1.1.6 Example XLFD Patterns for the Standard Names

Using the specified field values for these standard names, subsets of the standard names can be represented with various XLFD patterns. The XLFD pattern

```
-dt-application-*
```

logically matches the full set of XCDE Standard Application Font Names. (Note that no specific X server behaviour is implied). The pattern

```
-dt-application-bold-*-*-*-*-*-*-*-*-p-*-*-*-*
```

matches the bold, proportionally-spaced XCDE fonts, both serif and sans serif. And the pattern

```
-dt-application-*-*-*-*-*-*-*-*-*-m-*-*-*-*
```

matches the monospaced fonts (including both serif and sans serif).

The full set of XCDE Standard Application Font Names can be represented with the following patterns:

```
-dt-application-bold-i-normal-serif-*-*-*-*-p-*-iso8859-1
-dt-application-bold-r-normal-serif-*-*-*-*-p-*-iso8859-1
-dt-application-medium-i-normal-serif-*-*-*-*-p-*-iso8859-1
-dt-application-medium-r-normal-serif-*-*-*-*-p-*-iso8859-1
-dt-application-bold-i-normal-sans-*-*-*-*-p-*-iso8859-1
-dt-application-bold-r-normal-sans-*-*-*-*-p-*-iso8859-1
-dt-application-medium-i-normal-sans-*-*-*-*-p-*-iso8859-1
-dt-application-medium-r-normal-sans-*-*-*-*-p-*-iso8859-1
-dt-application-bold-i-normal-*-*-*-*-*-m-*-iso8859-1
-dt-application-bold-r-normal-*-*-*-*-*-m-*-iso8859-1
-dt-application-medium-i-normal-*-*-*-*-*-m-*-iso8859-1
-dt-application-medium-r-normal-*-*-*-*-*-m-*-iso8859-1
-dt-application-medium-r-normal-*-*-*-*-*-p-*-dtsymbol-1
```

Each of these 13 standard names comes in at least six point sizes.

19.1.1.7 Implementation of Font Names

The following requirements are placed on the implementation of the Standard Application Font Names:

- The names must be fully specified XLFD names, without wild cards.
- The WEIGHT_NAME, SLANT, SETWIDTH_NAME, SPACING, CHARSET_REGISTRY and CHARSET_ENCODING fields must contain valid values as defined previously and must match those in the underlying font.
- The ADD_STYLE_NAME field must contain either the **serif** or **sans** designation, whichever matches the underlying font.

19.1.1.8 Default XCDE Mappings for Latin-1 Locales

The default mapping of these standard application font names for the referenced ISO/IEC 8859-1:1987 standard locales is to the following standard X11R5 bitmapped fonts (the six minimum sizes are not shown explicitly in these patterns):

```
-adobe-times-bold-i-normal--*-*-*-*-p-*-iso8859-1
-adobe-times-bold-r-normal--*-*-*-*-p-*-iso8859-1
-adobe-times-medium-i-normal--*-*-*-*-p-*-iso8859-1
-adobe-times-medium-r-normal--*-*-*-*-p-*-iso8859-1
-adobe-helvetica-bold-o-normal--*-*-*-*-p-*-iso8859-1
-adobe-helvetica-bold-r-normal--*-*-*-*-p-*-iso8859-1
-adobe-helvetica-medium-o-normal--*-*-*-*-p-*-iso8859-1
-adobe-helvetica-medium-r-normal--*-*-*-*-p-*-iso8859-1
-adobe-courier-bold-o-normal--*-*-*-*-m-*-iso8859-1
-adobe-courier-bold-r-normal--*-*-*-*-m-*-iso8859-1
-adobe-courier-medium-o-normal--*-*-*-*-m-*-iso8859-1
-adobe-courier-medium-r-normal--*-*-*-*-m-*-iso8859-1
-adobe-symbol-medium-r-normal--*-*-*-*-p-*-adobe-fontspecific
```

A system may provide a different mapping of these standard names as long as all 13 names map to fonts of the appropriate design and style and the required six point sizes are available. The system documentation must document the system-specific default mapping for the standard names.

19.1.1.9 Font Names in app-defaults Files

An application can use a single *app-defaults* file to specify font resources and use it across all XCDE systems. Since most of the fields (FOUNDRY, FAMILY_NAME, WEIGHT_NAME, SLANT, SETWIDTH_NAME, ADD_STYLE_NAME, POINT_SIZE, SPACING, CHARSET_-REGISTRY and CHARSET_ENCODING) of the standard names are the same across different systems, these values can be used in the resource specification in the *app-defaults* file. However, other fields (PIXEL_SIZE, RESOLUTION_X, RESOLUTION_Y and AVERAGE_WIDTH) may vary across systems, and so must be wild-carded in the resource specification. For example:

```
appOne*headFont: -dt-application-bold-r-normal-sans- *-140-*-*-p-*-iso8859-1
appOne*linkFont: -dt-application-bold-i-normal-sans- *-100-*-*-p-*-iso8859-1
```

might be used to specify some of AppOne's default font resource needs.

19.1.1.10 Other Character Sets in the Common Locales

The standard application font names defined above are for use in locales using the referenced ISO / IEC 8859-1: 1987 standard character set only. For other locales supported by XCDE, there are no fonts guaranteed to be included. However, for the following locales, it is recommended that systems provide fonts with the following XLFD attribute values, and that they be accessible using these names.

Locales using ISO 8859-2, -3, -4, -5 (Cyrillic), -7 (Greek):
: The same values for FOUNDRY, FAMILY_NAME, WEIGHT_NAME, SLANT, SET_WIDTH, ADD_STYLE and SPACING as are used in this definition for the referenced ISO / IEC 8859-1: 1987 standard locale are recommended.

Japanese locales:
: Two values for the FAMILY_NAME attribute (**Gothic** and **Mincho**) and two values for the WEIGHT attribute (**medium** and **bold**) are recommended.

Chinese (Taiwan) locales:
: Two values for the FAMILY_NAME attribute (**Sung** and **Kai**) and two values for the WEIGHT attribute (**medium** and **bold**) are recommended.

Chinese (PRC) locales:
: Two values for the FAMILY_NAME attribute (**Song** and **Kai**) and two values for the WEIGHT attribute (**medium** and **bold**) are recommended.

Korean locales:
: Two values for the FAMILY_NAME attribute (**Totum** and **Pathang**) and two values for the WEIGHT attribute (**medium** and **bold**) are recommended. Note that these names are unofficial, tentative romanisations of the two common font families in use in Korea; Totum corresponds to fonts typically shipped as Gothic, Kodig or Dotum and Pathang corresponds to fonts typically shipped as Myungjo or Myeongjo. The official roman names for these fonts are under review and may be changed in the future by the Korean government, and thus may change for XCDE.

19.1.2 Standard Interface Font Names

The XCDE Standard Interface Font Names are a set of generic X Window System font names, needed by the XCDE GUI itself, that are used for user interface elements such as button labels, window titles and text fields. These names, for seven sizes of two typefaces, must exist on all XCDE systems. Seven sizes of a third typeface are recommended. They are typically mapped to existing fonts on the system using the font alias mechanism, although this method is not required.

The XCDE Standard Interface Font Names described here allow clients making up the XCDE desktop, such as *dtterm* and the window manager, a single set of default fonts in their *app-defaults* files, without concern for the system on which XCDE is running. (The XCDE Standard Application Font Names, described in Section 19.1.1 on page 343, provide a similar mechanism for applications running on the XCDE desktop.)

19.1.2.1 Background

Interface fonts are designed by user interface experts for the narrow purpose of making the menus, labels and fields of a graphical user interface highly readable. They are usually finely hand-tuned bitmapped fonts, intended for use on visual displays only and not on printers, and many of the glyphs have been specially modified for this purpose. Interface fonts can be contrasted with application fonts, which are the fonts used within an application running on the XCDE desktop. Interface fonts come in a restricted set of styles and are used for short strings of text, whereas application fonts usually come in a variety of designs, styles and weights and are used for emphasis, cross-references, section headers, and so forth.

19.1.2.2 Rationale

Common font names are required to prevent XCDE clients such as *dtterm* from needing different *app-defaults* files on each system.

Interface fonts are needed because of user interface and cognitive research that has examined the readability of various fonts on the display screens in use today and found that many fine adjustments (for example, for centering, baseline, height and alignment) must be made to characters in a font to make them clear, distinguishable and consistent when used for the interface objects of a GUI. And by using hand-tuned interface fonts for the GUI objects, the desktop can achieve a very clean, crisp visual appearance.

Interface fonts are broken into 2 categories: system and user. Cognitive research has shown that this distinction is important for the usability and readability of GUIs. System fonts are those used when the system is presenting information to the user (for example, in buttons). User fonts are those used for text that a user enters into the system (for example, for a text field or terminal emulator).

19.1.2.3 XLFD Field Values for the Standard Interface Font Names

These standard names are available using the X Window System XLFD font naming scheme. There are three aspects to the standard names:

- The *underlying font* on each system, to which a standard name is mapped, typically will be different on each system.
- The *standard name* itself, a full XLFD name mapped to the underlying font, may be different on each system in some of the XLFD fields. However, most of the fields are the same from system to system, allowing the *patterns* (described next) to be the same.

- The font resource *pattern* containing the * wildcards, used in *app-defaults* files, which will match the full XLFD name of the standard name, is the same across all systems, for a given use in an *app-defaults* file.

Systems must provide full XLFD names for the standard names, mapped to system-dependent underlying fonts, so that the XLFD patterns used in XCDE application *app-defaults* files will always match one of the full XLFD names provided.

The Standard Interface Font Names are identified by the presence of the following XLFD field name values:

- FOUNDRY is **dt**
- FAMILY_NAME is either **interface system** or **interface user** (there is a single space between the two words in each family name)

In addition, the other fields of the XLFD names defining the standard names are constrained as follows:

- WEIGHT_NAME is either **medium** or **bold**
- SLANT is always **r**
- SETWIDTH_NAME is always **normal**
- SPACING is **p** or **m** (it must be **m** for **interface user** fonts, and should be **p** for **interface system** fonts, although **m** is acceptable)
- ADD_STYLE_NAME contains both a nominal size value in the range **xxs** to **xxl** (see Section 19.1.2.5 on page 350), as well as either **sans** for sans serif fonts or **serif** for serif, if appropriate for the underlying font
- The numeric fields (PIXEL_SIZE, POINT_SIZE, RESOLUTION_X, RESOLUTION_Y, and AVERAGE_WIDTH) must contain the same values as the underlying font.
- CHARSET_REGISTRY and CHARSET_ENCODING are not specified; the standard names may be implemented for any XCDE locale.

Although the **sans** and **serif** values in the ADD_STYLE_NAME field are not required by the XLFD font convention, they are always part of the XCDE Standard Font Names when the underlying fonts are characterised as serif or sans serif. However, this document imposes no restriction on whether the interface fonts are serif or sans serif. The relevant attribute must be coded in the ADD_STYLE_NAME field. Thus, for example, the standard names for Japanese fonts, which are not characterised as being serif or sans serif, would not include this designation in the ADD_STYLE_NAME field.

19.1.2.4 Restricted Set of Styles Available

Unlike the Standard Application Font Names, only a limited set of styles is available in the Standard Interface Font Names. The styles available represent the minimum set currently considered necessary for the desktop GUI needs:

- a medium weight of an interface system font, preferably proportionally spaced (but mono-spaced is acceptable if appropriate for the locale)
- a medium weight of an interface user font, always mono-spaced
- a bold weight of an interface user font, always mono-spaced (the standard font names for this generic typeface are recommended if available for the targeted fonts and locale, but are not required).

19.1.2.5 Named Set of Point Sizes Available

In addition, the set of seven point sizes for each of the three styles that are part of this document are ''named'' point sizes, using string values in the ADD_STYLE_NAME field. Thus, XLFD patterns matching these names match a size based on the named size, not on a numeric size, even though the latter does exist in the XLFD name. These named sizes are used because the exact size of an interface font is less important than its nominal size, and implementation differences for the hand-tuned interface fonts do not allow common numeric point sizes to be assured across systems. The seven nominal sizes are as follows:

xxs extra extra small

xs extra small

s small

m medium

l large

xl extra large

xxl extra extra large

The goal of these named sizes is to provide enough fonts so that both the variety of display monitor sizes and resolutions that XCDE will run on, and the range of user preferences for comfortably reading button labels, window titles and so forth, can be accommodated in the GUI. Thus, both the smallest size, **xxs**, and the largest size, **xxl**, are meant to be reasonable sizes for displaying and viewing the XCDE desktop on common displays and X terminals; they are not meant to imply either hard-to-read fine print or headline-sized display type.

These named size values must occur first in the ADD_STYLE_NAME field, before any use of the values **serif** or **sans** (one of which is always required when the underlying font can be so characterised) and before any other additional stylistic attribute that might be appropriate. This is important when specifying wild-carded patterns in a resource specification for these fonts, since whether the underlying font these names are mapped to is serif or sans serif is not specified by XCDE, and the match must work for all XLFD names provided on XCDE systems.

19.1.2.6 Example XLFD Patterns for the Standard Names

Using these values, the XLFD pattern

```
-dt-interface*-*
```

logically matches the full set of XCDE Standard Interface Font Names. (Note that no specific X server behaviour is implied).

The full set of 21 XCDE Standard Interface Font Names can also be represented, in a more meaningful way, as follows:

```
-dt-interface system-medium-r-normal-*-*-*-*-*-*-*-*-iso8859-1
-dt-interface user-medium-r-normal-*-*-*-*-*-*-m-*-iso8859-1
-dt-interface user-bold-r-normal-*-*-*-*-*-*-m-*-iso8859-1
```

The full set of patterns, usable in *app-defaults* files, for all seven sizes for the system font, for example, is:

```
-dt-interface system-medium-r-normal-xxs*-*-*-*-*-*-*-*-iso8859-1
-dt-interface system-medium-r-normal-xs*-*-*-*-*-*-*-*-iso8859-1
-dt-interface system-medium-r-normal-s*-*-*-*-*-*-*-*-iso8859-1
-dt-interface system-medium-r-normal-m*-*-*-*-*-*-*-*-iso8859-1
-dt-interface system-medium-r-normal-l*-*-*-*-*-*-*-*-iso8859-1
-dt-interface system-medium-r-normal-xl*-*-*-*-*-*-*-*-iso8859-1
-dt-interface system-medium-r-normal-xxl*-*-*-*-*-*-*-*-iso8859-1
```

These patterns could be used in a resource file and will match the full XCDE Standard Interface Names for Latin-1 locales on all systems.

Note in these wild-carded XLFD names that the ADD_STYLE_NAME field has a pattern, such as **xxs***, and that the pattern is partly a string (**xxs**) and partly the pattern-matching character *. The full XLFD name this pattern matches—the XLFD name implementing the Standard Interface name—will often contain **sans** or **serif** in the field, after the **xxs** and a space, and so the * is essential to match that **sans** or **serif** string (and any additional style attribute string that might be in the underlying name). Note also that the SPACING field is wild-carded in the pattern for the system font, since either **p** or **m** may appear in the standard name being matched.

19.1.2.7 Implementation of Font Names

Each XCDE system provides mappings of its own fonts to XLFD names as described by this document. The actual XLFD names will vary from system to system, just as the fonts they are mapped to, since they contain some of the same values as the XLFD name of the underlying font.

There is no precise specification of how the named sizes **xxs** to **xxl** are mapped to sizes of underlying fonts in each system, although each size must be equal to or larger than the previous size. Nonetheless, some guidelines are appropriate.

Interface fonts have been developed because of human factors research on visual clarity of text on displays, and this has been done in the context of the display technology typically available today, mostly in the 100 dots per inch (DPI) range. That, and the use of standard point sizes (10, 12, 14, 18) in the graphics arts, have resulted in the development in the industry of hand-tuned bitmapped fonts for a set of ''pixel heights'' that are likely to be used for these standard names. However, making the XCDE desktop usable with a range of point sizes effectively means, in addition to legibility for the user, that the various XCDE applications fit ''appropriately'' on the screen using those point sizes. This means, for example, that two application windows can appear side by side on a typical display or that a certain number of buttons can appear across the screen.

Thus, these guidelines are expressed not only in pixel sizes, to reflect current usage, but also in percentage of monitor height. This allows them to remain appropriate as technological evolution improves display resolution and monitor size (for example, wall-mounted monitors). The ideal set of sizes would form a linear progression from the smallest (**xxs**) to the largest (**xxl**), although this is not achievable. The basic guideline is that the **xxs** font should be, in pixels, no less than 0.9% of the height of the display resolution, in pixels; the **xxl** font should be no more than 2.6% of the height.

As an approximate example that does not represent any existing mapping of fonts to a display, this table shows how the named sizes might map to real bitmapped fonts of a given pixel size, and how large those sizes are in percentage and point size terms:

Sample Range of Named Sizes on a 1280×1024 Display

named size	*number of pixels*	*size as % of 1024 height*	*point size on 100 DPI screen*
xxs	10	0.98%	7.2
xs	12	1.12%	8.7
s	14	1.37%	10.1
m	17	1.66%	12.3
l	20	1.95%	14.6
xl	23	2.25%	16.6
xxl	26	2.54%	18.8

Thus, the following requirements are placed on each implementation of the Standard Interface Font Names:

- The names must be fully specified XLFD names, without wild cards.
- The WEIGHT_NAME, SLANT, SETWIDTH_NAME, SPACING, CHARSET_REGISTRY and CHARSET_ENCODING fields must contain valid values as defined previously and must match those in the underlying font.
- The ADD_STYLE_NAME field must contain both a named size (for example, **xxs**) and, if appropriate, either the **serif** or **sans** designation, whichever matches the underlying font; any additional words about the style of the underlying font, if defined for the underlying font, must also be used. The named size must be first in the field, and must be separated from any following word in the field with a blank.
- The named sizes **xxs** through **xxl** must be mapped to fonts that are progressively larger than or equal to the previous one in the list. Thus, several standard names with adjacent sizes (for example, **xxs** and **xs**) may be mapped to the same font (for example, if there is not enough variety in sizes in the underlying fonts).
- The implemented names should attempt to meet the guidelines discussed in the previous paragraph and table.

For example, system A is assumed to be using the following sans serif font for the extra small system font:

```
-bitstream-swiss-medium-r-normal--11-90-85-85-p-81-iso8859-1
```

System B is using the following serif font for the extra small system font:

```
-vendorb-ersatz-medium-r-normal-Expert-8-80-75-75-m-72-iso8859-1
```

Their respective standard names would be implemented on their systems as:

```
-dt-interface system-medium-r-normal-xs sans-11-90-85-85-p-81-iso8859-1
-dt-interface system-medium-r-normal-xs serif Expert-8-80-75-75-m-72-iso8859-1
```

Defined this way, both names will match the single XLFD pattern used in a common *app-defaults* file:

```
-dt-interface system-medium-r-normal-xs*-*-*-*-*-*-*-*-iso8859-1
```

19.1.2.8 Default XCDE Mapping of the Standard Interface Font Names

There is no default mapping of these interface names to X11R5 fonts; the mapping is implementation-specific.

19.2 Icon Conventions

This section describes conventions for icon sizes, naming, location and usage within XCDE.

Both X Pixmap and X Bitmap icon file formats are used within the XCDE. X Pixmap (XPM) icons are multi-colour images based on the XPM format developed by Arnaud Le Hors. A pixmap file is a text file that can be read and modified by hand, in addition to using colour pixmap editors (like the XCDE Icon Editor). X Bitmap (XBM) icons are monochrome (two-colour) images based on the official X11 Bitmap File Format. A bitmap file is a text file that contains a binary representation of the bitmap, and as such is not easily read or modified by hand. The icon editor has the ability to write out any icon in either the XPM or XBM file format. For a more detailed description of these file formats and the icon editor, see the **XCDI** specification, **Section 3.1**, **XCDE Data Format Naming** and Chapter 10 on page 197.

19.2.1 File Naming

XCDE icon file names are typically in one of the following forms:

```
basename.format
basename.size.format
```

The *basename* is the logical name of the icon. The basenames for icons that are installed with XCDE begin with either **Dt** or **Fp**. **Dt** is the default prefix for all XCDE icons. The **Fp** prefix is used for icons that appear in the front panel when an icon other than the default **Dt** icon is desired. In this case, the basename is the same.

If an additional icon is needed for the client **iconImage** (iconified client window icon), a third prefix, **Ic**, is used.

The *format* is **pm** for a pixmap file and **bm** for a bitmap file. Size is a single letter: **l** for large, **m** for medium, **s** for small and **t** for tiny. Many of the logical icons are provided in multiple sizes for both colour and monochrome. This allows XCDE to use the optimal colour and size combination for the specific task and configuration the user is running. Many bitmap icons have a mask associated with the icon. These are named *basename.size*_**m.***format*. All icons are named so that the longest filename associated with that icon is 14 bytes or less; this allows it to be used on a short filename system. The longest filename can be described as **Dt***xxxxx*_**m.l.pm**, where *xxxxx* is the logical icon name.

Example

This is an example of icon files that might be associated with the icon **foo**. The single logical icon **foo** contains tiny, small, medium and large bitmap icons (with mask) and pixmap icons.

```
Dtfoo.t.pm
Dtfoo.t.bm
Dtfoo.t_m.bm
Dtfoo.s.pm
Dtfoo.s.bm
Dtfoo.s_m.bm
Dtfoo.m.pm
Dtfoo.m.bm
Dtfoo.m_m.bm
Dtfoo.l.pm
Dtfoo.l.bm
Dtfoo.l_m.bm
```

19.2.2 Icon Sizes

Icons of the following sizes and with the following suffixes are supported:

Large	48×48 icon with **.l** suffix
Medium	32×32 icon with **.m** suffix
Small	24×24 icon with **.s** suffix
Tiny	16×16 icon with **.t** suffix

The icon sizes used varies for different components, and is dependent on the display hardware.

19.2.3 Icon File Locations

XCDE has default locations where it looks to find system and user icon files. See Chapter 15 on page 323 for detailed information on where to install icons.

Chapter 20

Application Style Checklist

This chapter provides the list of style requirements for XCDE applications. XCDE requirements consist of the X/Open Motif requirements with XCDE-specific additions.

The checklist describes keys using a model keyboard mechanism. Wherever keyboard input is specified, the keys are indicated by the engravings that they have on the X/Open Motif model keyboard. Mouse buttons are described using a virtual button mechanism to better describe behaviour independent from the number of buttons on the mouse. For more information on the model keyboard and virtual button mechanisms, see the Preface and Section 2.2.1, ''Pointing Devices'' of the **OSF/Motif Style Guide**.

This checklist uses typographical conventions for keyboard and mouse inputs that differ from those used in the **OSF/Motif Style Guide**.

By default, this checklist assumes that the application is being designed for a left-to-right language environment in an English-language locale. Some sections of the checklist may require appropriate changes for other locales.

Each item in this checklist contains the corresponding section number from the checklist in the **OSF/Motif Style Guide**, if the item came from that list. Each item is also followed by a note containing a brief explanation or justification.

The headings used in this checklist correspond to the headings in the **OSF/Motif Style Guide** and the checklist items are labelled with the numbers used in that document. The XCDE-specific additions are labelled with alphabetic identifiers.

Each checklist item also has a priority label: Required, Recommended or Optional. The required items must be followed for an application to be XCDE compliant. Recommended items should be followed where feasible. Optional items are alternative implementations that the interface designer can choose.

20.1 Preface

1-1: [Required]

Each of the non-optional keys described on the X/Open Motif model keyboard is available either as specified or by using other keys or key combinations if the specified key is unavailable (Preface).

Note: The model keyboard does not correspond directly to any existing keyboard; rather, it assumes a keyboard with an ideal set of keys. However, to ensure consistency across applications, the non-optional keys or substitutes for them must always be available.

20.2 Input Models

20.2.1 Keyboard Focus Model

2-1: [Required]
Only one window at a time has the keyboard focus. The window that has the focus is highlighted. Within the window that has the keyboard focus, only one component at a time has the focus (Section 2.1 of the **OSF/Motif Style Guide**).

Note: The keyboard focus determines which component on the screen receives keyboard events. This requirement prevents confusion about which window and component have the focus.

2-2: [Required]
When the application uses an explicit focus policy, pressing BSelect does not move focus to a component that is not traversable or does not accept input (Section 2.1.2 of the **OSF/Motif Style Guide**).

Note: An explicit focus policy requires the user to explicitly select which window or component receives the keyboard focus. Generally, the user gives the focus to a window or component by pressing BSelect over it. However, this policy must not allow the user to give focus to a component that is not traversable or does not accept input.

2-3: [Required]
When the application uses an explicit focus policy, the component with the keyboard focus is highlighted by a location cursor (Section 2.1.2 of the **OSF/Motif Style Guide**).

Note: The user must know the location of the keyboard focus to be able to control an application.

20.2.2 Input Device Model

2-4: [Required]
The application supports methods of interaction for keyboard-only users. All features of the application are available from the keyboard (Section 2.2 of the **OSF/Motif Style Guide**).

Note: Not all users have access to a pointing device. These users must be able to access the full functionality of the application from the keyboard. Additionally, advanced users will be able to use the keyboard to perform some tasks more quickly than with a pointing device.

2-5: [Required]
The application uses the following bindings for mouse buttons (Section 2.2.1 of the **OSF/Motif Style Guide**):

BSelect [Optional]
This button should be used for selection, activation and setting the location cursor, and is the leftmost button, except for left-handed users, where it can be the rightmost button.

BTransfer [Optional]
This button should be used for moving and copying elements, and is mouse button 2, unless dragging is integrated with selection or the mouse has fewer than three buttons.

BMenu [Optional]
This button should be used for popping up menus, and is the rightmost button, except for left-handed users, where it can be the leftmost button or unless the mouse has fewer than three buttons. If the mouse has one button, BMenu is bound to Alt+BSelect.

Note: These bindings ensure a consistent interface for using standard mouse-based operations across applications.

2-6: [Required]
The application does not warp the pointer unless the application have given the user a means of disabling the behaviour (Section 2.2.4 of the **OSF/Motif Style Guide**).

Note: The pointer position is intended only as input to applications, not as an output mechanism. An application warps the pointer when it changes the pointer's position. This practice is confusing to users and reduces their sense of control over an application. Warping the pointer can also cause problems for users of absolute location pointing devices.

a: [Required]
Components and applications that have functions corresponding to the Motif-XCDE virtual keys must support those keys.

Note: If these virtual keys are available, the following mappings should be used. Priorities indicate the importance of implementing these functions in the application.

Help = F1 [Required]
Pressing the Help key must provide the user with help information in a window or in the status area.

Properties = Control+I [Required]
Pressing the Properties key must invoke a dialog box for making object-specific settings.

Undo = Control+Z [Required]
Pressing the Undo key must reverse the effect of the last applied operation. This is the primary key mapping for Undo.

Undo = Alt+Backspace [Optional]
This is a secondary key mapping for Undo. It should be supported in addition to Control+Z to help users migrating from previous versions of Motif, Microsoft Windows or OS/2.

Cut = Control+X [Required]
Pressing the Cut key must remove the selected object and places it in the clipboard. This is the primary key mapping for Cut.

Cut = Shift+Delete [Optional]
This is a secondary key mapping for Cut. It should be supported in addition to Control+X to help users migrating from previous versions of Motif, Microsoft Windows or OS/2.

Copy = Control+C [Required]
Pressing the Copy key must place a copy of the selected object in the clipboard. This is the primary key mapping for Copy.

Copy = Control+Insert [Optional]
This is a secondary key mapping for Copy. It should be supported in addition to Control+C to help users migrating from previous versions of Motif, Microsoft Windows or OS/2.

Paste = Control+V [Required]
Pressing the Paste key must place the contents of the clipboard at the selected location. This is the primary key mapping for Paste.

Paste = Shift+Insert [Optional]
This is a secondary key mapping for Paste. It should be supported in addition to Control+V to help users migrating from previous versions of Motif, Microsoft Windows or OS/2.

Open = Control+O [Optional]
Pressing the Open key opens the object, which is typically the default action.

Stop = Control+S [Optional]
Pressing the Stop key cancels an operation.

Again = Control+A [Optional]
Pressing the Again key repeats the last operation.

Print = Control+P [Optional]
Pressing the Print key initiates printing.

Save = Control+S [Optional]
Pressing the Save key saves the current file.

New = Control+N [Optional]
Pressing the New key creates a new object.

Bold = Control+B [Optional]
Pressing the Bold key makes the selected text bold.

Italic = Control+I [Optional]
Pressing the Italic key italicises the selected text.

Underline = Control+U [Optional]
Pressing the Underline key underlines the current text.

20.3 Navigation

20.3.1 Mouse-Based Navigation

3-1: [Required]
When the keyboard focus policy is explicit, pressing BSelect on a component moves focus to it, except for components, such as scroll bars, that are used to adjust the size and location of other elements (Section 3.1 of the **OSF/Motif Style Guide**).

Note: BSelect provides a convenient mechanism for using the mouse to move focus when the keyboard focus policy is explicit.

3-2: [Required]
When the pointer is on a menu, the application uses BSelect Press to activate the menu in a spring-loaded manner (Section 3.1 of the **OSF/Motif Style Guide**).

Note: A spring-loaded menu is one that appears when the user presses a mouse button, remains on the screen for as long as the button is pressed and disappears when the user releases the button. BSelect, mouse button 1, provides a means of activating spring-loaded menus that is consistent across applications.

3-3: [Required]
When the pointer is in an element with an inactive pop-up menu and the context of the element allows the pop-up menu to be displayed, the application uses BMenu Press to activate the pop-up menu in a spring-loaded manner (Section 3.1 of the **OSF/Motif Style Guide**).

Note: The availability of a pop-up menu can depend on the location of the pointer within an element, the contents of an element or the selection state of an element. BMenu, mouse button 3, provides a consistent means of activating a spring-loaded pop-up menu.

3-4: [Required]
If the user takes an action to post a pop-up menu and a menu can be posted for both an inner element and an outer element that contains the inner element, the pop-up menu for the internal element is posted (Section 3.1 of the **OSF/Motif Style Guide**).

Note: This requirement ensures that the pop-up menu for an internal element is always accessible.

3-5: [Required]
Once a pop-up menu is posted, BMenu behaves just as BSelect does for any menu system (Section 3.1 of the **OSF/Motif Style Guide**).

Note: The specified operation of BMenu is for manipulating pop-up menus.

3-6: [Required]
BSelect is also available from within posted pop-up menus and behaves just as in any menu system (Section 3.1 of the **OSF/Motif Style Guide**).

Note: Once a pop-up menu is posted, the user can select an element from it using the standard selection mechanism, BSelect.

3-7: [Required]
When a menu is popped up or pulled down in a posted manner, the application places the location cursor on the menu's default entry or on the first entry in the menu if there is no default entry (Section 3.1 of the **OSF/Motif Style Guide**).

Note: A posted menu remains visible until it is explicitly unposted. Placing the location cursor on the default entry allows the user to select the default operation easily. When there is no default entry, placing the location cursor on the first entry yields uniform behaviour across applications.

3-8: [Required]
The application removes a spring-loaded menu system when the mouse button that activated it is released, except when the button is released on a cascading button in the menu hierarchy (Section 3.1 of the **OSF/Motif Style Guide**).

Note: The concept of a spring-loaded menu system requires that the menu disappear when the mouse button is released.

3-9: [Required]
While a spring-loaded menu system is popped up or pulled down, moving the pointer within the menu system moves the location cursor to track the pointer (Section 3.1 of the **OSF/Motif Style Guide**).

Note: Once a spring-loaded menu system has appeared on the screen, the user must be able to maneuver the location cursor through the menu system using the mouse.

3-10: [Required]
When a spring-loaded menu system is popped up or pulled down and the pointer rests on a cascading button, the associated menu is pulled down and becomes traversable. The associated menu is removed, possibly after a short delay, when the pointer moves to a menu item outside of the menu or its cascading button (Section 3.1 of the **OSF/Motif Style Guide**).

Note: The user must be able to use the mouse to access all of the associated menus of a menu system. This feature allows the user to move quickly to any menu in a menu system.

3-11: [Required]
When a spring-loaded menu system that is part of the menu bar is pulled down, moving the pointer to any other element on the menu bar unposts the current menu system and posts the pull-down menu associated with the new element (Section 3.1 of the **OSF/Motif Style Guide**).

Note: This feature of a spring-loaded menu system allows the user to browse quickly through all of the menus attached to a menu bar.

3-12: [Required]
When a spring-loaded menu system is popped up or pulled down and the button that activated the menu system is released within a component in the menu system, that component is activated. If the release is on a cascading button or an option button, the associated menu is activated in a posted manner if it was not posted prior to the associated button press (Section 3.1 of the **OSF/Motif Style Guide**).

Note: Releasing the mouse button that activated a spring-loaded menu provides a means of activating a menu element that is consistent across applications.

3-13: [Required]
When the pointer is in an area with a pop-up menu, the application uses BMenu Click to activate the menu in a posted manner if it was not posted prior to the BMenu Click (Section 3.1 of the **OSF/Motif Style Guide**).

Note: BMenu Click provides a means of posting a pop-up menu that is consistent across applications.

3-14: [Required]
Once a pull-down or option menu is posted, BSelect Press in the menu system causes the menu to behave as a spring-loaded menu (Section 3.1 of the **OSF/Motif Style Guide**).

Note: This feature of a posted pull-down or option menu allows the user to switch easily between using a posted menu and a spring-loaded menu.

3-15: [Required]
If a button press unposts a menu and that button press is not also passed to the underlying component, subsequent events up to and including the button release are not passed to the underlying component (Section 3.1 of the **OSF/Motif Style Guide**).

Note: When a button press unposts a menu, the press can be passed to the underlying component. Whether or not it is passed to the underlying component, the press can have additional effects, such as raising and giving focus to the underlying window. If the press is not passed to the underlying component, events up to and including the release must not be passed to that component.

3-16: [Required]
Once a pop-up menu is posted, BSelect Press or BMenu Press in the menu system causes the menu to behave as a spring-loaded menu (Section 3.1 of the **OSF/Motif Style Guide**).

Note: This feature of a posted pop-up menu allows the user to switch easily between using a posted menu and a spring-loaded menu.

b: [Optional]
BMenu Press or BMenu Click on a menu bar item displays the menu.

c: [Required]
BMenu Press or BMenu Click on an option button displays the option menu.

d: [Required]
BSelect Press on a text entry field causes the text cursor to be inserted at the mouse cursor position.

20.3.2 Keyboard-Based Navigation

3-17: [Required]
In a text component, the text cursor is shown differently when the component does and does not have the keyboard focus (Section 3.2.1 of the **OSF/Motif Style Guide**).

Note: In a text component, the text cursor serves as the location cursor and, therefore, must indicate whether the component has keyboard focus.

3-18: [Required]
If a text component indicates that it has lost the keyboard focus by hiding the text cursor and if the component subsequently regains the focus, the cursor reappears at the same position it had when the component lost focus (Section 3.2.1 of the **OSF/Motif Style Guide**).

Note: To ensure predictability, the text cursor must not change position when a text component loses and then regains the keyboard focus.

3-19: [Required]
If a small component, such as a sash, indicates that it has the keyboard focus by filling, no other meaning is associated with the filled state (Section 3.2.1 of the **OSF/Motif Style Guide**).

Note: This requirement reduces possible confusion about the significance of filling in a small component.

3-20: [Required]
All components are designed and positioned within the application so that adding and removing each component's location cursor does not change the amount of space that the component takes up on the screen (Section 3.2.1 of the **OSF/Motif Style Guide**).

Note: For visual consistency, the sizes and positions of components should not change when keyboard focus moves from one component to another.

3-21: [Required]
Control+Tab moves the location cursor to the next field and Control+Shift+Tab moves the location cursor to the previous field. Unless Tab and Shift+Tab are used for internal navigation within a field, Tab also moves the location cursor to the next field and Shift+Tab also moves the location cursor to the previous field (Section 3.2.3 of the **OSF/Motif Style Guide**).

Note: These keys provide a consistent means of navigating among fields in a window.

3-22: [Required]
Tab (if not used for internal navigation) and Control+Tab move the location cursor forward through fields in a window according to the following requirements (Section 3.2.3 of the **OSF/Motif Style Guide**):

- If the next field is a control, Tab (if not used for internal navigation) and Control+Tab move the location cursor to that control.
- If the next field is a group, Tab (if not used for internal navigation) and Control+Tab move the location cursor to a traversable component within the group.
- If the next field contains no traversable components, Tab (if not used for internal navigation) and Control+Tab skip the field.

Note: These requirements ensure the consistent operation of Tab (if not used for internal navigation) and Control+Tab across applications.

3-23: [Required]
Shift+Tab (if not used for internal navigation) and Control+Shift+Tab move the location cursor backward through fields in the order opposite to that of Tab (if not used for internal navigation) and Control+Tab (Section 3.2.3 of the **OSF/Motif Style Guide**).

Note: These requirements result in the uniform operation of Shift+Tab (if not used for internal navigation) and Control+Shift+Tab across applications.

3-24: [Required]
When a window acquires focus, the location cursor is placed on the control that last had focus in the window, providing that all the following conditions are met (Section 3.2.3 of the **OSF/Motif Style Guide**):

- The window uses an explicit keyboard focus policy.
- The window acquires the focus through keyboard navigation or through a button press other than within the client area of the window.
- The window had the focus at some time in the past.
- The control that last had focus in the window is still traversable.

Note: This requirement ensures that when the user returns to a window after navigating away, the focus returns to the component where the user left it.

3-25: [Required]
Field navigation wraps between the first and last fields in the window. (Section 3.2.3 of the **OSF/Motif Style Guide**).

Note: This feature of field navigation provides the user with a convenient way to move through all of the fields in a window.

3-26: [Required]
When the Down Arrow and Up Arrow keys are used for component navigation within a field, they behave according to the following requirements (Section 3.2.3 of the **OSF/Motif Style Guide**):

In a left-to-right language environment, the Down Arrow key moves the location cursor through all traversable controls in the field, starting at the upper left and ending at the lower right, then wrapping to the upper left. If the controls are aligned in a matrix-like arrangement, Down Arrow first traverses one column from top to bottom, then traverses the column to its right and so on. In a right-to-left language environment, Down Arrow moves the location cursor through all traversable controls, starting at the upper right and ending at the lower left.

- Up Arrow moves the location cursor through all traversable controls in the field in the order opposite to that of Down Arrow.

Note: These requirements ensure a consistent means of navigating among components using the directional keys.

3-27: [Required]
When the Right Arrow and Left Arrow keys are used for component navigation within a field, they behave according to the following rules (Section 3.2.3 of the **OSF/Motif Style Guide**):

- In a left-to-right language environment, the Right Arrow moves the location cursor through all traversable controls in the field, starting at the upper left and ending at the lower right, then wrapping to the upper left. If the controls are aligned in a matrix-like arrangement, the Right Arrow first traverses one row from left to right, then traverses the row below it and so on. In a right-to-left language environment, the Right Arrow moves the location cursor through all traversable controls, starting at the lower left and ending at the upper right.
- Left Arrow moves the location cursor through all traversable controls in the field in the order opposite to that of the Right Arrow.

Note: These requirements ensure a consistent means of navigating among components using the directional keys.

3-28: [Required]
If a control uses the Right Arrow and Left Arrow for internal navigation, Begin moves the location cursor to the leftmost edge of the data or the leftmost element in

a left-to-right language environment. In a right-to-left language environment, Begin moves the location cursor to the rightmost edge of the data or the rightmost element (Section 3.2.3 of the **OSF/Motif Style Guide**).

Note: This requirement permits convenient navigation to the left or right edge of the data or the left or right element in a control.

3-29: [Required]
If a control uses the Right Arrow and Left Arrow keys for internal navigation, the End key moves the location cursor to the rightmost edge of the data or the rightmost element in a left-to-right language environment. In a right-to-left language environment, End moves the location cursor to the leftmost edge of the data or the leftmost element (Section 3.2.3 of the **OSF/Motif Style Guide**).

Note: This requirement permits convenient navigation to the left or right edge of the data or the left or right element in a control.

3-30: [Required]
If a control uses the Up Arrow and Down Arrow keys for internal navigation, Control+Begin moves the location cursor to one of the following (Section 3.2.3 of the **OSF/Motif Style Guide**):

- The first element
- The topmost edge of the data
- In a left-to-right language environment, the topmost left edge of the data; in a right-to-left language environment, the topmost right edge of the data

Note: This requirement permits convenient navigation to the beginning of the data in a control.

3-31: [Required]
If a control uses the Up Arrow and Down Arrow keys for internal navigation, Control+End moves the location cursor to one of the following (Section 3.2.3 of the **OSF/Motif Style Guide**):

- The last element
- The bottommost edge of the data
- In a left-to-right language environment, the bottommost right edge of the data; in a right-to-left language environment, the bottommost left edge of the data

Note: This requirement permits convenient navigation to the end of the data in a control.

e: [Optional]
Each time a new window is opened, keyboard focus is placed in the first field or location within the window or in a default location, if this is appropriate for the particular window.

f: [Required]
The Tab key moves input focus between push buttons within a group.

Note: The arrow keys also move the selected focus per the **OSF/Motif Style Guide**.

g: [Required]
The application uses the Control, Shift and Alt keys only to modify the function of other keys or key combinations.

h: [Optional]
The application should use the Alt key only to provide access to mnemonics.

20.3.3 Menu Traversal

3-32: [Required]
If the user traverses to a menu while the keyboard focus policy is implicit, the focus policy temporarily changes to explicit and reverts to implicit whenever the user traverses out of the menu system (Section 3.3 of the **OSF/Motif Style Guide**).

Note: Menus must always be traversable, even when the keyboard focus policy is generally implicit.

3-33: [Required]
The application uses the F10 key to activate the menu bar system if it is inactive. The location cursor is placed on the first traversable cascading button in the menu bar. If there are no traversable cascading buttons, the key does nothing (Section 3.3 of the **OSF/Motif Style Guide**).

Note: F10 provides a consistent means of traversing to the menu bar using the keyboard.

3-34: [Required]
When the keyboard focus is in an element with an inactive pop-up menu and the context of the element allows the pop-up menu to be displayed, the application uses the menu key to activate the pop-up menu. The location cursor is placed on the default item of the menu or on the first traversable item in the pop-up menu if there is no default item (Section 3.3 of the **OSF/Motif Style Guide**).

Note: The Menu key provides a uniform way of activating a pop-up menu from the keyboard.

3-35: [Required]
When the keyboard focus is in an option button, the application uses the Select key or the Spacebar to post the option menu. The location cursor is placed on the previously selected item in the option menu; or, if the option menu has been pulled down for the first time, the location cursor is placed on the default item in the menu. If there is an active option menu, the Return, Select or Spacebar keys select the current item in the option menu, unpost the menu system and return the location cursor to the option button (Section 3.3 of the **OSF/Motif Style Guide**).

Note: These keys provide a means of posting an option menu from the keyboard that is consistent across applications.

3-36: [Required]
The application uses the Down Arrow, Left Arrow, Right Arrow and Up Arrow keys to traverse through the items in a menu system (Section 3.3 of the **OSF/Motif Style Guide**).

Note: The Down Arrow, Left Arrow, Right Arrow and Up Arrow directional keys provide a consistent means of navigating among items in a menu system.

3-37: [Required]
When a menu traversal action traverses to the next or previous component in a menu or menu bar, the order of traversal and the wrapping behaviour are the same as that of the corresponding component navigation action within a field, as described in Section 3.2.3 (Section 3.3 of the **OSF/Motif Style Guide**).

Note: This requirement provides consistency between menu traversal and component navigation within a field.

3-38: [Required]
If the application uses any two-dimensional menus, they do not contain any cascading buttons (Section 3.3 of the **OSF/Motif Style Guide**).

Note: Cascading buttons in a two-dimensional menu would restrict the user's ability to navigate to all of the elements of the menu using the keyboard.

3-39: [Required]
When focus is on a component in a menu or menu bar system, the Down Arrow key behaves according to the following rules (Section 3.3 of the **OSF/Motif Style Guide**):

- If the component is in a vertical or two-dimensional menu, traverse down to the next traversable component, wrapping within the menu if necessary.
- If the component is in a menu bar and the component with the keyboard focus is a cascading button, post its associated pull-down menu and traverse to the default entry in the menu or, if the menu has no default, to the first traversable entry in the menu.

Note: This requirement results in consistent operation of the directional keys in a menu or menu bar system.

3-40: [Required]
When focus is on a component in a menu or menu bar system, the Up Arrow key behaves according to the following rules (Section 3.3 of the **OSF/Motif Style Guide**):

- If the component is in a vertical or two-dimensional menu, this action traverses up to the previous traversable component, wrapping within the menu if necessary and proceeding in the order opposite to that of the Down Arrow key.

Note: This requirement results in consistent operation of the directional keys in a menu or menu bar system.

3-41: [Required]
When focus is on a component in a menu or menu bar system, the Left Arrow key behaves according to the following rules (Section 3.3 of the **OSF/Motif Style Guide**):

- If the component is in a menu bar or two-dimensional menu, but not at the left edge, traverse left to the previous traversable component.
- If the component is at the left edge of a menu bar, wrap within the menu bar.
- If the component is at the left edge of a vertical or two-dimensional menu that is the child of a vertical or two-dimensional menu, unpost the current menu and traverse to the parent cascading button.
- If the component is at the left edge of a vertical or two-dimensional menu that is the child of a menu bar, unpost the current menu and traverse left to the previous traversable entry in the menu bar. If that entry is a cascading button, post its associated pull-down menu and traverse to the default entry in the menu or, if the menu has no default, to the first traversable entry in the menu.

Note: This requirement results in consistent operation of the directional keys in a menu or menu bar system.

3-42: [Required]
When focus is on a component in a menu or menu bar system, the Right Arrow key behaves according to the following rules (Section 3.3 of the **OSF/Motif Style Guide**):

- If the component is a cascading button in a vertical menu, post its associated pull-down menu and traverse to the default entry in the menu or, if the menu has no default, to the first traversable entry in the menu.
- If the component is in a menu bar or two-dimensional menu, but not at the right edge, traverse right to the next traversable component.
- If the component is at the right edge of a menu bar, wrap within the menu bar.
- If the component is not a cascading button and is at the right edge of a vertical or two-dimensional menu and if the current menu has an ancestor cascading button (typically in a menu bar) from which the Down Arrow key posts its associated pull-down menu, unpost the menu system pulled down from the nearest such ancestor cascading button and traverse right from that cascading button to the next traversable component. If that component is a cascading button, post its associated pull-down menu and traverse to the default entry in the menu or, if the menu has no default, to the first traversable entry in the menu.

Note: This requirement results in consistent operation of the directional keys in a menu or menu bar system.

3-43: [Required]
All menu traversal actions, with the exception of menu posting, traverse to tear-off buttons in the same way as for other menu entries (Section 3.3 of the **OSF/Motif Style Guide**).

Note: Traversal of tear-off buttons must be consistent with traversal of other menu items.

3-44: [Required]
If the application uses the F10, Menu or Cancel key to unpost an entire menu system and an explicit focus policy is in use, the location cursor is moved back to the component that had it before the menu system was posted (Section 3.3 of the **OSF/Motif Style Guide**).

Note: Returning the location cursor to the component that had it previously allows the user to resume a task without disruption.

20.3.4 Scrollable Component Navigation

3-45: [Required]
Any scrollable components within the application support the appropriate navigation and scrolling operations. The application uses the page navigation keys Page Up, Page Down, Control+Page Up (for Page Left) and Control+Page Down (for Page Right) for scrolling the visible region by a page increment (Section 3.4 of the **OSF/Motif Style Guide**).

Note: A user must be able to view and access the entire contents of a scrollable component.

3-46: [Required]
When scrolling by a page, the application leaves at least one unit of overlap between the old and new pages (Section 3.4 of the **OSF/Motif Style Guide**).

Note: The overlap between one page and the next yields visual continuity for the user.

3-47: [Required]
Any keyboard operation that moves the cursor to or in the component or that inserts, deletes or modifies items at the cursor location scrolls the component so that the cursor is visible when the operation is complete (Section 3.4 of the **OSF/Motif Style Guide**).

Note: The user must be able to see the results of moving the location cursor or operating on the contents of the scrollable component.

3-48: [Required]
If a mouse-based scrolling action is in progress, the Cancel key cancels the scrolling action and returns the scrolling device to its state prior to the start of the scrolling operation (Section 3.4 of the **OSF/Motif Style Guide**).

Note: The Cancel key provides a convenient way for the user to cancel a scrolling operation.

20.4 Selection

20.4.1 Selection Models

4-1: [Required]
The system supports five selection models: single selection, browse selection, multiple selection, range selection and discontiguous selection (Section 4.1 of the **OSF/Motif Style Guide**).

Each collection has one or more appropriate selection models. The model limits the kinds of choices the user can make in the collection. Some collections enforce a selection model, while others allow the user or application to change it.

20.4.1.1 Mouse-Based Single Selection

4-2: [Required]
In a collection that uses single selection, when BSelect is clicked in a deselected element, the location cursor moves to that element, that element is selected and any other selection in the collection is deselected (Section 4.1.1 of the **OSF/Motif Style Guide**).

Note: Single selection is the simplest selection model, used to select a single element. BSelect, mouse button 1, provides a consistent means of selecting an object within a group using the mouse.

20.4.1.2 Mouse-Based Browse Selection

4-3: [Required]

In a collection that uses browse selection, when BSelect is released in a selectable element, that element is selected and any other selection in the collection is deselected. As BSelect is dragged through selectable elements, each element under the pointer is selected and the previously selected element is deselected. The selection remains on the element where BSelect is released and the location cursor is moved there (Section 4.1.2 of the **OSF/Motif Style Guide**).

Note: Browse selection is used to select a single element. It also allows the user to browse through the collection by dragging BSelect. See Section 20.4.1.3.

4-4: This item of the **OSF/Motif Style Guide** is not applicable.

20.4.1.3 Mouse-Based Multiple Selection

i: [Required]

If the application contains collections that follow the multiple selection model, BAdjust is supported and behaves equivalent to BSelect, when the BTransfer button is currently configured to behave as BAdjust.

Note: On a three-button mouse, button 2 is typically used for the BTransfer (or BSelect) function. However, in a XCDE environment, the user may change an environment setting indicating that mouse button 2 should be used for the BAdjust function instead. BAdjust can be used to toggle the selection state of elements under the multiple selection model.

j: [Required]

In a collection that uses multiple selection, clicking BSelect or BAdjust on an unselected element adds that element to the current selection. Clicking BSelect or BAdjust on a selected element removes that element from the current selection. Clicking BSelect or BAdjust moves the location cursor to that element.

20.4.1.4 Mouse-Based Range Selection

4-5: [Required]

This item of the **OSF/Motif Style Guide** has been replaced by items **k** and **l**.

k: [Required]

In a collection that uses range selection, pressing BSelect on an unselected element sets an anchor on the element or at the position where BSelect was pressed and deselects all elements in the collection. If BSelect is released before the drag threshold has been exceeded, then the element under the pointer is selected. If BSelect Motion exceeds the drag threshold, then a new selection begins. The anchor and the current position of the pointer determine the current range. As BSelect is dragged through the collection, the current range is highlighted. When BSelect is released, the anchor does not move and all the elements within the current range are selected (Section 4.1.4 of the **OSF/Motif Style Guide**).

Note: Range selection allows the user to select multiple contiguous elements of a collection by pressing and dragging BSelect.

l: [Required]

In a collection that uses range selection, pressing BSelect on an currently selected element causes none of the other elements in the selection set to be deselected. If BSelect is released before the drag threshold is exceeded, then, at that point, all other

elements are deselected and the element under the pointer remains selected. If BSelect Motion exceeds the drag threshold, then no element is deselected and a drag operation begins.

4-6: [Required]
In a text-like collection that uses range selection, the anchor point is the text pointer position when BSelect is pressed and the current range consists of all elements between the anchor point and the current text pointer position (Section 4.1.4 of the **OSF/Motif Style Guide**).

Note: In text-like collections, elements are ordered linearly and a text pointer is always considered to be between elements at a point near the actual pointer position.

4-7: [Required]
In a graphics-like or list-like collection that uses a marquee to indicate the range of a range selection, the current range consists of those elements that fall completely within the marquee. If there is an anchor element, the marquee is always made large enough to enclose it completely. Otherwise, an anchor point is used and is the point at which BSelect was pressed; the anchor point determines one corner of the marquee. If the collection is not arranged as a list or matrix, the marquee is extended to the pointer position. If the collection is arranged as a list or matrix, the marquee is either extended to completely enclose the element under the pointer or extended to the pointer position. Clicking BSelect on a selectable element makes it an anchor element, selects it and deselects all other elements (Section 4.1.4 of the **OSF/Motif Style Guide**).

Note: A marquee or highlighted rectangle, is often used to indicate the range of a selection in graphics-like and list-like collections.

4-8: This item of the **OSF/Motif Style Guide** is not applicable.

m: [Required]
If the application contains collections that follow the range selection model, BAdjust is supported and behaves equivalent to Shift+BSelect, when the BTransfer button is currently configured to behave as BAdjust.

Note: On a three-button mouse, button 2 is typically used for the BTransfer function. However, in XCDE, the user may change an environment setting indicating that mouse button 2 should be used for the BAdjust function instead. BAdjust can be used to extend the selection set in the same manner as Shift+BSelect.

n: [Required]
In a collection that uses range selection, when the user presses Shift+BSelect or BAdjust, the anchor remains unchanged and an extended range for the selection is determined, based on one of the extension models:

Reselect [Optional]
The extended range is determined by the anchor and the current pointer position, in the same manner as when the selection was initially made.

Enlarge Only [Optional]
The selection can only be enlarged. The extended range is determined by the anchor and the current pointer position, but then is enlarged to include the current selection.

Balance Beam [Optional]

A balance point is defined at the midpoint of the current selection. When the user presses Shift+BSelect or BAdjust on the opposite side of the balance point from the anchor, this model works equivalent to the reselect model. When the user presses Shift+BSelect, BAdjust or starts a navigation action modified by Shift on the same side of the balance point as the anchor, this model moves the anchor to the opposite end of the selection and then works equivalent to the reselect model.

Note: When the user releases BSelect or BAdjust, the anchor does not move, all the elements within the extended range are selected and all the elements outside of it are deselected.

20.4.1.5 Mouse-Based Discontiguous Selection

4-9: [Required]

In a collection that uses discontiguous selection, the behaviour of BSelect is equivalent to the range selection model. After the user sets the anchor with BSelect, Shift+BSelect is equivalent to the range selection model (Section 4.1.5 of the **OSF/Motif Style Guide**).

Note: Discontiguous selection is an extension of range selection that allows the user to select multiple discontiguous ranges of elements.

4-10: [Required]

In a collection that uses discontiguous selection, when the current selection is not empty and the user clicks Control+BSelect, the anchor and location cursor move to that point. If the current selection is not empty and the user clicks Control+BSelect on an element, the selection state of that element is toggled and that element becomes the anchor element (Section 4.1.5 of the **OSF/Motif Style Guide**).

Note: In discontiguous selection, Control+BSelect Click provides a convenient means of moving the anchor and toggling the selection state of the element under the pointer.

4-11: [Required]

In a collection that uses discontiguous selection, Control+BSelect Motion toggles the selection state of a range of elements. The range itself is determined as for BSelect Motion. Releasing Control+BSelect toggles the selection state of the elements in the range according to one of two models (Section 4.1.5 of the **OSF/Motif Style Guide**):

Anchor Toggle [Optional]

Toggling is based on an anchor element. If the range is anchored by a point and is not empty, the anchor element is set to the element within the range that is nearest to the anchor point. Toggling sets the selection state of all elements in the range to the inverse of the initial state of the anchor element.

Full Toggle [Optional]

The selection state of each element in the extended range is toggled.

Note: In discontiguous selection, Control+BSelect provides a convenient means of toggling the selection state of elements in a range.

4-12: [Required]
In a collection that uses discontiguous selection, after Control+BSelect toggles a selection, Shift+BSelect or Control+Shift+BSelect extends the range of toggled elements. The extended range is determined in the same way as when Shift BSelect is used to extend a range selection. When the user releases Control+Shift+BSelect, the selection state of elements added to the range is determined by the toggle model in use (either Anchor Toggle or Full Toggle). If elements are removed from the range, they either revert to their state prior to the last use of Control+BSelect or change to the state opposite that of the elements remaining within the extended range (Section 4.1.5 of the **OSF/Motif Style Guide**).

Note: Shift+BSelect and Control+Shift+BSelect provide a convenient means of extending the range of toggled elements.

o: [Required]
In a collection that uses discontiguous selection, BAdjust can be used to extend the range of a discontiguous selection. The extended range is determined in the same way as when BAdjust is used to extend a range selection.

Note: On a three-button mouse, the button 2 is typically used for the BTransfer function. However, in a XCDE environment, the user may change an environment setting indicating that mouse button 2 should be used for the BAdjust function instead. BAdjust can be used to extend the selection set in the same manner as Shift+BSelect.

20.4.1.6 Keyboard Selection

4-13: [Required]
The selection models support keyboard selection modes according to the following rules (Section 4.1.6 of the **OSF/Motif Style Guide**):

- Single selection supports only add mode.
- Browse selection supports only normal mode.
- Multiple selection supports only add mode.
- Range selection supports normal mode. If it also supports add mode, normal mode is the default.
- Discontiguous selection supports both normal mode and add mode. Normal mode is the default.

Note: Selection must be available from the keyboard. In normal mode, used for making simple contiguous selections from the keyboard, the location cursor is never disjoint from the current selection. In add mode, used for making more complex and possibly disjoint selections, the location cursor can move independently of the current selection.

4-14: [Required]
If a collection supports both normal mode and add mode, Shift+F8 switches from one mode to the other. Mouse-based selection does not change when the keyboard selection mode changes. In editable components, add mode is a temporary mode that is exited when the user performs an operation on the selection or deselects the selection (Section 4.1.6 of the **OSF/Motif Style Guide**).

Note: Shift+F8 provides a convenient means of switching between normal mode and add mode.

Keyboard-Based Single Selection

4-15: [Required]
In a collection that uses single selection, the navigation keys move the location cursor independently from the selected element. If the user presses the Select key or the Spacebar on an unselected element, the element with the location cursor is selected and any other selection in the collection is deselected (Section 4.1.6.1 of the **OSF/Motif Style Guide**).

Note: Single selection supports only add mode. Pressing the Select key or the Spacebar is similar to clicking BSelect.

Keyboard-Based Browse Selection

4-16: [Required]
In a collection that uses browse selection, the navigation keys move the location cursor and select the cursored element, deselecting any other element. If the application has deselected all elements or if the cursor is left disjoint from the selection, the Select key or the Spacebar selects the cursored element and deselects any other element (Section 4.1.6.2 of the **OSF/Motif Style Guide**).

Note: Browse selection supports only normal mode. A navigation operation is similar to dragging BSelect.

Keyboard-Based Multiple Selection

4-17: [Required]
In a collection that uses multiple selection, the navigation keys move the location cursor independently from the current selection. The Select key or the Spacebar on an unselected element adds the element to the current selection. Pressing the Select key or the Spacebar on a selected element removes the element from the current selection (Section 4.1.6.3 of the **OSF/Motif Style Guide**).

Note: Multiple selection supports only add mode. Pressing the Select key or the Spacebar is similar to clicking BSelect.

Keyboard-Based Range Selection

4-18: [Required]
In a collection that uses range selection and is in normal mode, the navigation keys move the location cursor and deselect the current selection. If the cursor is on an element, it is selected. The anchor moves with the location cursor.

Note: Text-like collections can use a different model in which the navigation keys leave the anchor at its current location, except that, if the current selection is not empty, it is deselected and the anchor is moved to the location of the cursor prior to navigation (Section 4.1.6.4 of the **OSF/Motif Style Guide**).

Range selection supports normal mode and, if the collection also supports add mode, normal mode is the default.

4-19: [Required]
In a collection that uses range selection, whether in normal mode or add mode, the Select key or Spacebar (except in a text component) moves the anchor to the cursor, deselects the current selection and, if the cursor is on an element, selects the element. Unless the anchor is on a deselected item, Shift+Select or Shift+Spacebar

(except in text) extends the selection from the anchor to the cursor, based on the extension model used by Shift+BSelect (Reselect, Enlarge Only or Balance Beam) (Section 4.1.6.4 of the **OSF/Motif Style Guide**).

Note: In range selection, pressing the Select key or Spacebar is similar to clicking BSelect and pressing Shift+Select or Shift+Spacebar extends the range as with Shift+BSelect.

4-20: [Required]
In a collection that uses range selection and is in normal mode, using Shift in conjunction with the navigation keys extends the selection, based on the extension model used by Shift+BSelect. If the current selection is empty, the anchor is first moved to the cursor. The cursor is then moved according to the navigation keys and the selection is extended based on the extension model used by Shift+BSelect (Section 4.1.6.4 of the **OSF/Motif Style Guide**).

Note: In range selection, shifted navigation extends the selection in a similar manner to dragging Shift+BSelect.

4-21: [Required]
In a collection that uses range selection and is in add mode, the navigation keys move the location cursor, but leave the anchor unchanged. Shifted navigation moves the location cursor according to the navigation keys and the selection is extended based on the extension model used by Shift+BSelect (Section 4.1.6.4 of the **OSF/Motif Style Guide**).

Note: Shifted navigation in add mode is similar to shifted navigation in normal mode, except that when the selection is empty the anchor does not move to the cursor prior to navigation.

Keyboard-Based Discontiguous Selection

4-22: [Required]
In a collection that uses discontiguous selection and is in normal mode, all keyboard operations have the same effect as in the range selection model (Section 4.1.6.5 of the **OSF/Motif Style Guide**).

Note: Normal mode does not permit multiple discontiguous selections.

4-23: [Required]
In a collection that uses discontiguous selection and is in add mode, the Select key or Spacebar moves the anchor to the location cursor and initiates toggling. If the cursor is on an element, the selection state of that element is toggled, but the selection state of all other elements remains unchanged. Shift+Select or Shift+Spacebar and shifted navigation operations extend the selection between the anchor and the location cursor, based on the toggle mechanism used by Control+BSelect (Anchor Toggle or Full Toggle) (Section 4.1.6.5 of the **OSF/Motif Style Guide**).

Note: Add mode permits use of the keyboard to make multiple discontiguous selections.

20.4.1.7 Canceling a Selection

4-24: [Required]
The application uses the Cancel key to cancel or undo any incomplete motion operation used for selection. Once the user presses the Cancel key to cancel a motion operation, the application ignores subsequent key and button releases until after all buttons and keys are released. Pressing the Cancel key while extending or toggling leaves the selection state of all elements as they were prior to the button press (Section 4.1.7 of the **OSF/Motif Style Guide**).

Note: The Cancel key allows the user to cancel an incomplete selection operation quickly and consistently.

20.4.1.8 Autoscrolling and Selection

4-25: [Required]
If the user drags the pointer out of a scrollable collection during a motion-based selection operation, autoscrolling is used to scroll the collection in the direction of the pointer. If the user presses the Cancel key with BSelect pressed, the selection operation is canceled as described in Section 4.1.7 (Section 4.1.8 of the **OSF/Motif Style Guide**).

Note: Autoscrolling provides a convenient means of extending a selection to elements outside the viewport of a scrollable collection.

20.4.1.9 Selecting and Deselecting All Elements

4-26: [Required]
In a collection that uses multiple, range or discontiguous selection, Control+/ selects all the elements in the collection, places the anchor at the beginning of the collection and leaves the location cursor at its previous position (Section 4.1.9 of the **OSF/Motif Style Guide**).

Note: Control+/ provides the user with a convenient means of selecting all of the objects in a collection.

4-27: [Required]
In a collection that is in add mode, Control+\ deselects all the elements in the collection. In a collection that is in normal mode, Control+\ deselects all the elements in the collection, except the element with the location cursor if the location cursor is being displayed. In either mode, Control+\ leaves the location cursor at its current position and moves the anchor to the location cursor (Section 4.1.9 of the **OSF/Motif Style Guide**).

Note: Control+\ allows the user to deselect all of the selected objects quickly and uniformly.

20.4.1.10 Using Mnemonics for Elements

4-28: [Required]
If the application supports mnemonics associated with selectable elements, typing a mnemonic while the collection has the keyboard focus is equivalent to moving the location cursor to the element and pressing the Select key or Spacebar (Section 4.1.10 of the **OSF/Motif Style Guide**).

Note: Mnemonics within a collection of selectable elements provide an additional selection method.

20.4.2 Selection Actions

4-29: [Required]
When the keyboard focus policy is explicit, the destination component is the editable component that last had the keyboard focus. When the keyboard focus policy is implicit, the destination component is the editable component that last received mouse button or keyboard input (Section 4.2.1 of the **OSF/Motif Style Guide**).

Note: The destination component is used to identify the component on which certain operations, primarily data transfer operations, act. There is only one destination component at a time.

4-30: [Required]
If the keyboard focus is in a component (or a pop-up menu of a component) that supports selections, operations that act on a selection act on the selection in that component (Section 4.2.2 of the **OSF/Motif Style Guide**).

Note: A selection operation acts on the component that has focus, if that component supports selections.

4-31: [Required]
If the keyboard focus is in a component (or a pop-up menu of a component) that supports some operation that does not act on a selection, invoking the operation acts on that component (Section 4.2.2 of the **OSF/Motif Style Guide**).

Note: An operation that does not act on a selection acts on the component that has focus, if that component supports the operation.

4-32: [Required]
Inserting or pasting elements into a selection, except for a primary transfer operation at the bounds of the primary selection, first deletes the selection if pending delete is enabled (Section 4.2.3 of the **OSF/Motif Style Guide**).

Note: Pending delete controls the conditions under which the selection is deleted. It is enabled by default.

4-33: [Required]
In normal mode, inserting or pasting elements disjoint from the selection also deselects the selection, except for primary transfer operations whose source and destination are in the same collection. In add mode, the selection is not deselected (Section 4.2.3 of the **OSF/Motif Style Guide**).

Note: In add mode, a transfer operation that is disjoint from the selection does not affect the selection.

4-34: [Required]
In editable list-like and graphics-like collections, Delete deletes the selected elements (Section 4.2.3 of the **OSF/Motif Style Guide**).

Note: Delete provides a consistent means of deleting the selection.

4-35: [Required]
In editable text-like collections, Delete and Backspace behave as follows:

- If the selection is not empty and the control is in normal mode, the selection is deleted.
- If the selection is not empty, the control is in add mode and the cursor is not disjoint from the selection, the selection is deleted.

- If the selection is not empty and the control is in add mode, but the cursor is disjoint from the selection, Delete deletes one character forward and Backspace deletes one character backward.
- If the selection is empty, Delete deletes one character forward and Backspace deletes one character backward.

Note: In text, Delete and Backspace provide a convenient way to delete the entire selection or single characters.

20.4.3 Transfer Models

4-36: [Required]
If the move, copy or link operation the user requests is not available, the transfer operation fails (Section 4.3 of the **OSF/Motif Style Guide**).

Note: Three transfer operations are generally available: copy, move and link. The user requests one of these operations by pressing the buttons or keys appropriate for the type of transfer. In general, for mouse-based operations, the modifier Control forces a copy, Shift forces a move and Control+Shift forces a link. However, any requested transfer operation must fail if that operation is not available.

4-37: [Required]
If a collection does not have a fixed insertion point or keep elements ordered in a specific way, the insertion position for transferred data is determined as follows (Section 4.3 of the **OSF/Motif Style Guide**):

- For BTransfer-based (or BSelect) primary and drag transfer operations, excepted as noted below for text collections, the insertion position is the position at which the user releases BTransfer (or BSelect).
- In a text-like collection, when the user drops selected text, the insertion position is the position at which the user releases BTransfer (or Bselect). When the user drops an icon, the insertion position is the text cursor and the data is pasted before it.
- In a list-like collection, the insertion position for other transfer operations is the element with the location cursor and the data is pasted before it.

Note: The insertion position is the position in the destination where transferred data is placed. Some mouse-based transfer operations place data at the pointer position if possible. Other operations, including keyboard-based transfer, generally place the data at the location cursor.

p: [Required]
The application supports the use of mouse button 1 to perform drag-and-drop operations.

Note: In X/Open Motif, drag and drop is typically performed using button 2 on a three-button mouse (BTransfer). However, in XCDE, mouse button 1 (BSelect) should be supported for drag and drop to support mouse usage compatible with other graphical user interface (GUI) environments. A drag can be initiated with either mouse button 1 or mouse button 2.

q: [Required]
When button 2 of a three-button mouse is configured to operate as BAdjust, the application does not perform any BTransfer (or BSelect) operations when clicking mouse button 2.

Note: On a three-button mouse, button 2 is typically used for the BTransfer function. However, in a XCDE environment, the user can change an environment setting indicating that mouse button 2 should be used for the BAdjust function instead. When this is the case, BAdjust click should not result in the transfer of any data.

r: [Required]
BSelect should always initiate a drag if the drag is started on a selected item. The drag starts once the drag threshold has been reached. This is true for text regions, scrolling lists and other similar elements.

20.4.3.1 Clipboard Transfer

4-38: [Required]
Keyboard-based clipboard selection actions are available in every editable collection in the application (Section 4.3.1 of the **OSF/Motif Style Guide**).

Note: Clipboard selection actions must be available from the keyboard.

4-39: [Required]
The application uses the Cut key (or Shift+Delete) and the Cut entry on the Edit menu to cut the selected elements from an editable component to the clipboard (Section 4.3.1 of the **OSF/Motif Style Guide**).

Note: The Cut key (or Shift+Delete) and the Cut entry on the Edit menu offer a consistent means of cutting the selection to the clipboard from the keyboard.

4-40: [Required]
The application uses the Copy key (or Control+Insert) and the Copy entry on the Edit menu to copy the selected elements to the clipboard (Section 4.3.1 of the **OSF/Motif Style Guide**).

Note: The Copy key (or Control+Insert) and the Copy entry on the Edit menu offer a consistent means of copying the selection to the clipboard from the keyboard.

4-41: [Required]
The application uses the Paste key (or Shift+Insert) to paste the contents of the clipboard into an editable component (Section 4.3.1 of the **OSF/Motif Style Guide**).

Note: The Paste key (or Shift+Insert) offers a consistent way of pasting the contents of the clipboard from the keyboard.

4-42: [Required]
If Paste or Paste Link is invoked using a component's pop-up menu, the data is pasted at the insertion position of the component. However, if the pop-up menu is popped up over a selection, the selection is first deleted, even if pending delete is disabled and the pasted data replaces it, if possible (Section 4.3.1 of the **OSF/Motif Style Guide**).

Note: Popping up a pop-up menu over a selection indicates that a Paste or Paste Link operation should replace the selection.

4-43: [Required]
If Paste or Paste Link is invoked from the Edit menu or by a keyboard operation and the insertion position in the target component is not disjoint from a selection, the pasted data replaces the selection contents if pending delete is enabled (Section 4.3.1 of the **OSF/Motif Style Guide**).

Note: Pending delete determines whether the selection is deleted when the insertion position is not disjoint from the selection and Paste or Paste Link is invoked from the Edit menu or by a keyboard operation.

20.4.3.2 Primary Transfer

4-44: [Required]
In an editable collection, BTransfer Click, Control+BTransfer Click, Alt, Copy and Control+Alt+Insert copy the primary selection to the insertion position, as defined in Section 4.3 of the **OSF/Motif Style Guide**. (Note that the insertion position is usually different for mouse and keyboard operations.) (Section 4.3.2 of the **OSF/Motif Style Guide**)

Note: These operations provide a convenient way for the user to force a copy operation.

4-45: [Required]
In an editable collection, Shift+BTransfer Click, Alt+Cut and Alt+Shift+Delete move the primary selection to the insertion position, as defined in Section 4.3 of the **OSF/Motif Style Guide**. (Note that the insertion position is usually different for mouse and keyboard operations.) (Section 4.3.2 of the **OSF/Motif Style Guide**)

Note: These operations provide a convenient way for the user to force a move operation.

4-46: [Required]
In an editable collection, Control+Shift+BTransfer Click places a link to the primary selection at the insertion position, as defined in Section 4.3 (Section 4.3.2 of the **OSF/Motif Style Guide**).

Note: Control+Shift+BTransfer provides a convenient way for the user to force a link operation.

4-47: [Required]
A Primary Move moves the primary selection as well as the elements selected; that is, the element moved to the destination becomes selected as the primary selection. Primary Copy and Primary Link do not select transferred data at the destination (Section 4.3.2 of the **OSF/Motif Style Guide**).

Note: This requirement provides the expected treatment of the selection in a move, copy and link operation.

20.4.3.3 Quick Transfer

4-48: [Required]
All text components support quick transfer (Section 4.3.3 of the **OSF/Motif Style Guide**).

Note: Quick transfer is used to make a temporary selection and then immediately move, copy or link that selection to the insertion position of the destination component. In text, quick transfer provides a convenient way to move, copy or link text without disturbing the primary selection.

4-49: [Required]
If a component supports quick transfer, Alt+BTransfer Motion or Control+Alt+BTransfer Motion temporarily selects elements in the specified range and, on release, copies them to the insertion position of the destination component (Section 4.3.3 of the **OSF/Motif Style Guide**).

Note: These operations provide a convenient way to perform a quick copy.

4-50: [Required]
If a component supports quick transfer, Alt+Shift+BTransfer Motion temporarily selects elements in the specified range and, on release, moves them to the insertion position of the destination component (Section 4.3.3 of the **OSF/Motif Style Guide**).

Note: This operation provides a convenient way to perform a quick cut.

4-51: [Required]
If a component supports quick transfer, Control+Alt+Shift+BTransfer Motion temporarily selects elements in the specified range and, on release, places a link to them at the insertion position of the destination component (Section 4.3.3 of the **OSF/Motif Style Guide**).

Note: This operation provides a convenient way to perform a quick link.

4-52: [Required]
Quick transfer does not disturb the primary selection or affect the clipboard, except when the destination of the transfer is within or on the boundaries of the primary selection and pending delete is enabled. In this case, quick transfer deletes the contents of the primary selection, leaving an empty primary selection, before pasting the transferred elements (Section 4.3.3 of the **OSF/Motif Style Guide**).

Note: Quick transfer is a secondary selection mechanism, so it cannot disrupt the primary selection. When the destination of the transfer is in the primary selection, quick transfer replaces the primary selection with the secondary selection.

4-53: [Required]
With quick transfer, the range of the temporary selection is determined by using the same model as when BSelect Motion determines the range of a primary selection (Section 4.3.3 of the **OSF/Motif Style Guide**).

Note: This requirement provides consistency between primary selection and quick transfer operations.

4-54: [Required]
If the user drags the pointer out of a scrollable collection while making the temporary selection, autoscrolling is used to scroll the collection in the direction of the pointer. If the user releases BTransfer with the pointer outside of the collection or if the user presses the Cancel key with BTransfer pressed, the highlighting is removed and a transfer is not performed (Section 4.3.3 of the **OSF/Motif Style Guide**).

Note: Autoscrolling provides a convenient means of extending a temporary selection to elements outside the viewport of a scrollable collection.

20.4.3.4 Drag Transfer

4-55: [Required]
In a collection that supports selection, Shift+BTransfer Release or Shift+BSelect Release forces a drag move operation. If a move is not possible, the operation fails (Section 4.3.4 of the **OSF/Motif Style Guide**).

Note: This mechanism offers a convenient way for the user to force a move operation.

4-56: [Required]
In a collection that supports selection, Control+BTransfer Release or Shift+BSelect Release forces a drag copy operation. If a copy is not possible, the operation fails (Section 4.3.4 of the **OSF/Motif Style Guide**).

Note: This mechanism offers a convenient way for the user to force a copy operation.

4-57: [Required]
In a collection that supports selection, Control+Shift+BTransfer Release or Shift+BSelect Release forces a drag link operation. If a link is not possible, the operation fails (Section 4.3.4 of the **OSF/Motif Style Guide**).

Note: This mechanism offers a convenient way for the user to force a link operation.

4-58: [Required]
When a drag move operation moves a selection within the same component, the selection moves along with the elements selected (Section 4.3.4 of the **OSF/Motif Style Guide**).

Note: In other words, when selected elements are moved with a drag operation, they should stay selected after the move. This mechanism offers a convenient way to move the selection within a component.

4-59: [Required]
In text-like collections, initiating a drag within a selected region drags the entire text selection (Sections 4.3.4 and 4.3.5 of the **OSF/Motif Style Guide**).

Note: To be consistent, drag-and-drop actions must operate on the entire selection.

4-60: [Required]
In list-like and graphics-like collections, initiating a drag on a selected element drags the entire selection (Sections 4.3.4 and 4.3.5 of the **OSF/Motif Style Guide**).

Note: To be consistent, drag-and-drop actions must operate on the entire selection.

4-61: [Required]
In list-like and graphics-like collections, initiating a drag with BTransfer or BSelect on an unselected element drags just that element and leaves the selection unaffected (Section 4.3.4 of the **OSF/Motif Style Guide**).

Note: Unselected elements can be dragged without affecting the selection.

4-62: [Required]
When a drag is initiated in an unselected region and the pointer is over two possible draggable elements, the drag uses the draggable element highest in the stacking order (Section 4.3.4 of the **OSF/Motif Style Guide**).

Note: This requirement ensures the consistency of drag operations.

4-63: [Required]
When the application starts a drag operation, the pointer is replaced with a drag icon (Section 4.3.4.1 of the **OSF/Motif Style Guide**).

Note: A drag icon provides visual feedback that a drag operation is in progress.

4-64: [Required]
All drag icons used by the application include a source indicator (Section 4.3.4.1 of the **OSF/Motif Style Guide**).

Note: A source indicator gives a visual representation of the elements being dragged.

4-65: [Required]
Pressing the Cancel key ends a drag-and-drop operation by canceling the drag in progress (Section 4.3.4.2 of the **OSF/Motif Style Guide**).

Note: The Cancel key provides a consistent way for the user to cancel a drag operation.

4-66: [Required]
Releasing BTransfer ends a drag-and-drop operation (4.3.4.3 of the **OSF/Motif Style Guide**).

Note: Releasing BTransfer offers a consistent means of ending a drag operation.

4-67: [Required]
When BTransfer (or BSelect) is released, the drop operation ordinarily occurs at the location of the hot spot of the drag icon pointer and into the highest drop zone in the stacking order. However, if a drop occurs within a selection and pending delete is enabled, the transferred data replaces the contents of the entire selection (Section 4.3.4.3 of the **OSF/Motif Style Guide**).

Note: This requirement provides consistency in the treatment of mouse-based transfer operations.

4-68: [Required]
After a successful transfer, the data is placed in the drop zone and any transfer icon used by the application is removed (Section 4.3.4.4 of the **OSF/Motif Style Guide**).

Note: A transfer icon can be used to represent the type of data being transferred during a drop operation. A successful drop operation results in the transfer of data.

4-69: [Required]
After a failed transfer, the data remains at the drag source and is not placed in the drop zone. Any transfer icon used by the application is removed (Section 4.3.4.4 of the **OSF/Motif Style Guide**).

Note: A failed drop operation does not result in the transfer of data.

s: [Recommended]
In a collection that supports selection, if BTransfer Motion (or BSelect Motion) results in the start of a drag operation, feedback should be presented to the user that indicates that a copy, move or link operation is in progress. Whether the operation is a copy, move or link depends on the type of object created at the drop zone and whether the source object is removed.

Note: Although an unmodified drag typically results in a move operation, depending on the location of the source object and the target drop zone, the drag may in fact result in a copy or link operation. For example, dragging an icon representing an attachment to a mail message typically results in a copy of the attachment being created as opposed to the original being removed from the mail message. Any feedback presented should incorporate use of a drag icon that portrays the source object being manipulated.

t: [Recommended]

In a collection that supports selection, if Control+BTransfer Motion or Control+BSelect Motion results in the start of a drag operation, feedback should be presented to the user that indicates that a copy operation is in progress.

Note: The feedback presented should incorporate use of a drag icon that portrays the source object being copied.

u: [Recommended]

In a collection that supports selection, if Control+Shift+BTransfer Motion or Control+Shift+BSelect Motion results in the start of a drag operation, feedback should be presented to the user that indicates that a link operation is in progress.

Note: The feedback presented should incorporate use of a drag icon that portrays the source object being linked.

v: [Recommended]

In a collection that supports copy, move or link operations that can be performed by dragging, the feedback presented to the user during the drag operation should indicate whether a single object or multiple objects are being manipulated.

Note: Feedback provided during the drag operation should ensure that the user feels confident that the desired set of objects is being dragged. The drag icon used for multi-object drag operations should integrate the feedback used to indicate whether the operation is a move, copy or link.

w: [Optional]

If the application allows the user to paste data into its data pane, it allows the user to drag and drop files from the File Manager into the data pane.

Note: The user should be able to drag and drop files into application data panes. The result should be the inclusion of some element of the file or the display of an error message indicating that the file selected cannot be incorporated into the application's data. Drag transfers that are accepted can result in a number of different responses from the application:

1. The icon image for the file might be inserted at the drop point
2. The application might perform some activity using the data contained within the file as its input
3. The data contained within the file might be inserted at the drop point
4. The name of the file might be inserted at the drop point

20.5 Component Activation

20.5.1 Basic Activation

5-1: [Required]
The application uses BSelect to activate a button (Section 5.1 of the **OSF/Motif Style Guide**).

Note: BSelect, mouse button 1, provides a consistent means of activating a button using the mouse.

5-2: [Required]
When a button has the focus, the application uses the Select key or Spacebar to activate the button (Section 5.1 of the **OSF/Motif Style Guide**).

Note: The Select key and Spacebar provide a uniform way of selecting a button. Selecting a button is equivalent to activating the button.

5-3: [Required]
When an activatable menu entry has the focus, the application uses the Select, Spacebar, Enter or Return key to activate the entry (Section 5.1 of the **OSF/Motif Style Guide**).

Note: The Select, Spacebar, Enter and Return keys offer a consistent means of activating a menu entry using the keyboard.

5-4: [Required]
When BSelect is pressed over a button, the appearance of the button changes to indicate that releasing BSelect will activate the button. If, while BSelect is pressed, the pointer is moved outside of the button, the visual state is restored. If, while BSelect is still pressed, the pointer is moved back inside of the button, the visual state is again changed to indicate the pending activation. If BSelect is pressed and released within a button, the button is activated, regardless of whether the pointer has moved out of the button while it was pressed (Section 5.1 of the **OSF/Motif Style Guide**).

Note: The visual state of a button offers a cue to the user about whether or not the button will be activated when the mouse button is released.

5-5: [Required]
If a selectable element of a collection is activatable, BSelect Click, the Select key and Spacebar (except in text) select it. BSelect Click 2 selects and activates it (Section 5.1 of the **OSF/Motif Style Guide**).

Note: This requirement provides for consistent integration of activation and selection in a collection where elements can be both selected and activated.

x: [Required]
The time allowed to detect a double click (*doubleClickTime: 500) is no less than 500 milliseconds.

20.5.2 Accelerators

5-6: [Required]

If the application uses accelerators, the component with the accelerator displays the accelerator key or key combination following the label of the component (Section 5.2 of the **OSF/Motif Style Guide**).

Note: An accelerator is a key or key combination that invokes the action of some component regardless of the position of the location cursor when the accelerator is pressed. So that the user knows that there is an accelerator associated with a component, the accelerator must be displayed.

5-7: [Required]

If a button with an accelerator is within a primary or secondary window or within a pull-down menu system from its menu bar, it is activatable whenever the input focus is in the window or the menu bar system. If a button with an accelerator is within a pop-up menu system, it is activatable whenever the focus is in the pop-up menu system or the component with the pop-up menu (Section 5.2 of the **OSF/Motif Style Guide**).

Note: An accelerator must be able to be activated from the window or component associated with the accelerator.

20.5.3 Mnemonics

5-8: [Required]

If the application uses mnemonics, the label for the component with the mnemonic contains the character that is its mnemonic. If the label does not naturally contain the character, the mnemonic is placed in parentheses following the label (Section 5.3 of the **OSF/Motif Style Guide**).

Note: A mnemonic is a single character that can be associated with any component that contains a text label. Mnemonics provide a fast way of selecting a component from the keyboard. To let the user know that there is a mnemonic associated with a selection, the mnemonic is underlined in the label of the selection by the toolkit. For a mnemonic to be underlined, the label for a selection must contain the mnemonic character. If the label does not contain the mnemonic, putting the mnemonic in parentheses following the label provides visual consistency.

y: [Required]

Mnemonic characters are chosen for ease-of-location within the text of a label. The following rules are applied in sequence, selecting the first possible match that results in a unique mnemonic:

1. The first character of the label is used.
2. If there is more than one word in the label, the first character of the second word is used.
3. The last character of the label is used.
4. The first unique character in the label from the second character on is used.

5-9: [Required]

All mnemonics are case insensitive for activation (Section 5.3 of the **OSF/Motif Style Guide**).

Note: The user must be able to activate a mnemonic by pressing either the lowercase or the uppercase variant of the mnemonic key.

5-10: [Required]
When the location cursor is within a menu or a menu bar, pressing the mnemonic key of a component within that menu or menu bar moves the location cursor to the component and activates it. If a mnemonic is used for an option button or for a cascading button in a menu bar, pressing Alt and the mnemonic anywhere in the window or its menus moves the cursor to the component with that mnemonic and activates it (Section 5.3 of the **OSF/Motif Style Guide**).

Note: A mnemonic is generally activatable when the location cursor is within the component that contains the mnemonic. Pressing Alt and the mnemonic provides a way to activate a visible mnemonic when the location cursor is within the window that contains the mnemonic.

20.5.4 Tear-Off Activation

5-11: [Required]
Activating a tear-off button tears off the menu that contains the button (Section 5.4 of the **OSF/Motif Style Guide**).

Note: A tear-off button is similar to a push button with the special interaction of tearing off the menu from its cascading button. Tear-off buttons use the same basic activation as other buttons.

5-12: [Required]
When a menu with a tear-off button is posted, pressing BTransfer in the tear-off button starts a tear-off action. As long as BTransfer is held, a representation of the menu follows the movement of the pointer. Releasing BTransfer ends the tear-off action by unposting the menu system, creating a new window at the current pointer location that contains the contents of the menu and giving focus to the new window in explicit pointer mode (Section 5.4 of the **OSF/Motif Style Guide**).

20.5.5 Help Activation

5-13: [Required]
The application uses the Help key on a component to invoke any context-sensitive help for the component or its nearest ancestor with context-sensitive help available (Section 5.5 of the **OSF/Motif Style Guide**).

Note: The Help key offers the user a consistent mechanism for invoking context-sensitive help.

z: [Required]
The application provides context-sensitive help at all locations.

Note: The user must never get a ‘‘help not available’’ message.

20.5.6 Default Activation

5-14: [Required]
If the application uses default push buttons in a window, the current default push button is highlighted. When the focus is on a push button, its action is the default action and the push button shows default highlighting. If the default action in a window varies, some push button always has default highlighting, except when there is no current default action (Section 5.6 of the **OSF/Motif Style Guide**).

Note: Placing emphasis on the default push button in a dialog box provides the user with a visual cue about the expected reply to the dialog box.

5-15: [Required]
When focus is in a window with a default action and an activatable menu does not have the focus, the Enter key and Control+Return invoke the default action. If focus is in a component other than multi-line text or an activated menu, Return also invokes the default action. These actions have no other effect on the component with the focus, unless the default action has some effect on that component (Section 5.6 of the **OSF/Motif Style Guide**).

Note: These requirements ensure that the means of invoking a default action are consistent across applications.

5-16: [Required]
Except in the middle of a button motion operation, pressing the Cancel key anywhere in a dialog box is equivalent to activating the Cancel push button in the dialog box (Section 5.6 of the **OSF/Motif Style Guide**).

Note: The Cancel key provides a uniform means of canceling dialog box from the keyboard.

20.5.7 Expert Activation

5-17: [Required]
If the application supports expert activation, expert actions exist only as shortcuts to application features that are available through another mechanism (Section 5.7 of the **OSF/Motif Style Guide**).

Note: Expert activation, using mouse double-clicking on buttons, provides a convenient way for experienced users to perform certain tasks quickly. However, new users and keyboard-only users must be able to perform the same tasks.

5-18: [Required]
When the focus is on a button used for expert activation, no default action is available, unless the default and expert actions are the same (Section 5.7 of the **OSF/Motif Style Guide**).

Note: This requirement minimises possible confusion between default and expert activation.

5-19: [Required]
If a component with an expert action is selectable, activating the expert action first selects the component and then performs the expert action (Section 5.7 of the **OSF/Motif Style Guide**).

Note: A user must be able to select a component, even if it has an expert action associated with it.

20.5.8 Previewing and Autorepeat

5-20: [Required]
If the application supports activation preview using BSelect, the previewing information is removed when the user releases BSelect (Section 5.8 of the **OSF/Motif Style Guide**).

Note: Activation preview presents the user with additional information that describes the effect of activating a button. This information cannot interfere with the normal operation of the application.

20.5.9 Cancel Activation

5-21: [Required]
Pressing the Cancel key stops current interaction in the following contexts (Section 5.9 of the **OSF/Motif Style Guide**):

- During a mouse-based selection or drag operation, it cancels the operation.
- During a mouse-based scrolling operation, it cancels the scrolling action and returns the system to its state prior to the start of the scrolling operation.
- Anywhere in a dialog box that has a Cancel push button, it is equivalent to activating that push button, except during a mouse-based selection or drag operation.
- In a pull-down menu, it either dismisses the menu and moves the location cursor to the cascading button used to pull it down or unposts the entire menu system. In a pop-up menu, option menu, tear-off menu or menu bar, it unposts the menu system.
- When the focus is in a torn off menu window, it closes the torn off menu window.

Note: These requirements for the Cancel key ensure the consistent operation of the key across applications.

20.6 Window Management

20.6.1 Window Support

This section corresponds to section 7.2 of the **OSF/Motif Style Guide**. The different window types are discussed throughout the **OSF/Motif Style Guide** and this chapter. In particular, see Section 20.7 on page 393.

aa: [Required]
Application windows are clearly distinguishable as primary or secondary windows based on appearance and behaviour.

Primary Window:

- Primary window decoration (see Section 20.6.2 on page 389)
- Primary window management (see Section 20.6.5.3 on page 391)
- Window stacking, workspace placement and minimisation can be independent of other primary windows

Secondary Window:

- Secondary window decoration (see Section 20.6.2)
- Secondary window management (see Section 20.6.5.3 on page 391)
- Window stacking, workspace placement and minimisation tied to the associated primary window

20.6.2 Window Decorations

ab: [Required]
Windows that support particular window management functionality must request the corresponding window decoration (for example, a window that can be minimised must request the minimise button).

ac: [Required]
Windows that support any window management functionality (move, resize, minimise, maximise, close and others) have a window menu with items for that functionality.

ad: [Required]
The application follows XCDE window decoration conventions, as shown in the following table.

XCDE Window Decoration Conventions

	Border	Title	Menu	Min	Max	Resize
Primary Window:						
Default	Yes	Yes	Yes	Yes	Yes[1]	Yes[1]
Front Panel	Yes[2]	No	Yes[2]	Yes	No	No
Secondary Window:						
Default	Yes	Yes	Yes	No	No[3]	No[3]
Front Panel	No	Yes	Yes	No	No	No

1. Decorations for resize and maximise are provided for primary windows, if appropriate.
2. The Front Panel has custom visuals for the window decorations.
3. Secondary windows are designed such that resizing and maximisation are not necessary or appropriate. If a secondary window must be resizable and maximisable, the associated decorations are displayed.

ae: [Required]
The application follows XCDE window menu conventions. Items appear in the window menu if they are applicable to the window or its minimised window icon.

- Restore (R)
- Move (M)
- Size (S)
- Minimize (n)
- Maximize (x)

- Lower (L)
- Occupy Workspace ... (O)
- Occupy All Workspaces (A)
- Unoccupy Workspace (U)
- Close (C) Alt+F4

af: [Optional]
The application should not add items to the window menu. If an extraordinary requirement has an application add items to the window menu, the items should be appended to the end of the menu with a separator between Close and the application items.

ag: [Optional]
Accelerators, other than Alt+F4 for Close, should not be used in the window menu.

Note: This minimises conflict with other uses of the Alt key for application accelerators, localisation and others.

20.6.3 Window Navigation

This section corresponds to section 7.4 of **OSF/Motif Style Guide**. There are no checklist items for application developers.

20.6.4 Icons

ah: [Optional]
The application should provide unique window icons for their primary windows. The window icon image should have a similar appearance to the associated file or Front Panel icon image.

ai: [Optional]
The window icon label should contain the same text as the title of the corresponding primary window or an abbreviated form of it. See Section 20.7.1 on page 393 for window title guidelines.

aj: [Optional]
The window icon image should have a similar appearance to the associated file or Front Panel icon image.

20.6.5 Application Window Management

20.6.5.1 Window Placement

ak: [Recommended]
The application should not require or force windows or window icons to be positioned at a particular screen location.

al: [Recommended]
The application should place a secondary window relative to the associated primary window. It should be placed close to, but not obscuring, the component that caused it to be displayed and the information that is necessary to interact with the dialog box.

am: [Optional]
Some suggestions are given in section 6.2.4.3, ‘‘Determining Dialog Box Location

and Size,'' of the **OSF/Motif Style Guide**. Additional or modified recommendations include:

If a dialog box does not relate to specific items in the underlying window, it should be placed below the menu bar (if there is one) and centered (horizontally) over the work area.

an: [Recommended]
If a secondary window is allowed to be stacked below its associated primary window (not constrained to stay on top of the primary window), it should be placed such that it is not completely covered by the primary window. This recommendation takes precedence over other placement recommendations.

Note: This recommendation is included to accommodate certain legacy applications and is applicable only when the user has selected this behaviour. Newly written XCDE applications must not rely on this type of window placement.

ao: [Recommended]
If a menu or dialog box is already on display, reinvoking the command that caused it to be displayed should automatically bring that window or menu to the front of the window stack without changing its position on the screen.

20.6.5.2 Window (Document) Clustering

ap: [Optional]
Windows that are closely related in supporting a particular task should be placed in a window cluster. Secondary windows are automatically placed in a window cluster with the associated primary window. Windows in a window cluster are stacked together, minimised or normalised together and kept in the same workspace.

Note: Currently the only mechanism for forming a window cluster that is supported by the window manager is to indicate a primary-secondary relationship.

20.6.5.3 Window Management Actions

aq: [Required]
Windows should follow XCDE window management functionality conventions, as shown in the following table.

XCDE Window Management Conventions

	Close	Move	Lower	Min	Max	Resize
Primary Window:						
Default	Yes	Yes	Yes	Yes	Yes[1]	Yes[1]
Front Panel	No	Yes	Yes	Yes	No	No
Secondary Window:						
Default	Yes	Yes	Yes	No	No[2]	No[2]
Front Panel	Yes	Yes	Yes	No	No	No

1. Resize and maximise functionality should be provided for primary windows if appropriate.

2. Secondary windows can contain the Maximize and Resize window manager functions, if appropriate.

ar: [Required]
Windows that support particular window management functionality should request corresponding window decoration (for example, a window that can be minimised should request the minimise button).

as: [Required]
Windows that have form factor constraints set window manager hints for minimum size, maximum size, aspect ratio and resize increment as appropriate.

at: [Recommended]
Maximising a window should show more content (objects or controls) if appropriate (as opposed to scaling up the sizes of objects and controls).

au: [Required]
Windows that have Close or Exit functionality support the window management protocol for Close if there is a window menu. In the case of dialog boxes, the Close item on the window menu corresponds to the Cancel functionality or dialog box dismissal with no further action taken.

av: [Recommended]
When the application creates a new window, it should come up in the user's current workspace and only occupy that single workspace.

aw: [Recommended]
Application windows that are related to a particular task should move together between workspaces.

20.6.6 Session Management Support

ax: [Required]
The application should support Interclient Communications Conventions Manual (ICCCM—see the X/Open CAE Specification, **Window Management: File Formats and Application Conventions**) mechanisms for session management of their primary windows and key properties.

ay: [Required]
The application should support ICCCM mechanisms for session management of all associated windows (that is, secondary windows that may include help windows).

az: [Optional]
The application should accept messages from the XCDE Session Manager that inform it the user is logging out and should save the application state at that time.

ba: [Optional]
An application that has a single primary window that is open at the time the user logs out should restore the primary window, in the workspace last occupied, when the user logs in again.

bb: [Optional]
Save user context wherever possible. For example, an application that supports the editing of files should save the state of the file at logout and should restore the file in the application window when users log in again.

bc: [Optional]
An application that has multiple primary windows that are open at the time the

user logs out should restore all primary windows, in their respective workspaces, when the user logs in again.

20.7 Application Design Principles

20.7.1 Layout

20.7.1.1 Main Window

6-1: [Required]
The application should be composed of at least one main window (Section 6.2.1.1 of the **OSF/Motif Style Guide**). A main window contains a client area and, optionally, a menu bar, a command area, a message area and scroll bars. The client area contains the framework of the application.

Note: The use of a main window ensures interapplication consistency.

bd: [Required]
The default size of the application's main window is large enough to accommodate a typical amount of data, but does not fill the entire physical display size (and thus minimises visual conflicts with other applications).

Note: Each application potentially must share the display with other applications. The default window size should not take up all the available screen space.

be: [Required]
Resize corners should be included in any main window that incorporates a scrolling data pane or list.

Note: Any changes to the overall size of the window should result in a corresponding increase or decrease in the size of the scrollable portion. Additionally, the application might reorganise elements within the window based on the increased or decreased amount of space (for example, it might reorganise a row of buttons into two rows).

6-2: [Required]
If the application has multiple main windows that serve the same primary function, each window closes and iconifies separately (Section 6.2.1.1 of the **OSF/Motif Style Guide**).

Note: For example, a text editor might allow the user to edit multiple documents, each in its own main window. Each window is then treated as a separate application and can be closed or iconified when it is not being used.

6-3: [Required]
If the application has multiple main windows that serve different primary functions, each window should be able to iconify independently of the other windows (Section 6.2.1.1 of the **OSF/Motif Style Guide**).

Note: For example, a debugger might provide separate main windows for editing source code, examining data values and viewing results. Each window can be iconified when it is not being used, but it is up to the application to decide whether each window closes separately or whether closing one window closes the entire application.

20.7.1.2 Window Titles

bf: [Optional]
The title of the primary window (the main window the application displays to the user) should be the name of the application.

Note: The title does not have to be the actual name of the executable invoked by the user.

The application developer should carefully consider how the title chosen for the primary window works when it is used in icons and pop-up windows. If the name of the pop-up window is too long, the application may remove the application title, but without the title, users might have difficulty telling which pop-up windows belong with the originating primary window.

bg: [Optional]
The application should use initial capital letters for each word in the title (in languages that support capitalisation).

bh: [Optional]
The application should follow the application name for each property window, as a minimum, with the title Properties and the name of the object it affects.

bi: [Optional]
The title of each pop-up window should begin with the application title followed by a colon, then the title of the pop-up window. The colon should have a space both before and after it for readability.

Pop-up windows should always indicate which primary window they are associated with (which primary window invoked that pop-up).

bj: [Optional]
When the application has files that can be loaded or saved, the application should use a hyphen to denote the current file name. The hyphen should have a space before and after it. Only the base name of the file should be displayed, not the entire pathname.

Note: The hyphen is used to denote specific instances of a window or data. The colon serves to delimit general categories or commands. For example, a file manager might have the following title for a Properties dialog box:

```
File Manager : Properties - myfile
```

bk: [Recommended]
The application should follow the application name for each command window with the same title that is on the window button or window item users choose to display that window.

bl: [Optional]
In the case of multiple primary windows, the application name should be included at the beginning of each window title, with an added name that uniquely identifies that primary window. No separator should be provided for these names (for example, Calendar Manager Multi-Browse, Catalogue Search, Admintool Databases).

bm: [Optional]
An abbreviated name for the application may be used on other windows, so long as it is done on all windows.

20.7.1.3 Menu Bar

The following requirements apply only in a left-to-right language environment in an English-language locale. The application must make the appropriate changes for other locales.

6-4: [Required]
If the application has a menu bar, it is a horizontal bar at the top edge of the application, just below the title area of the window frame.

Note: A menu bar organizes the most common features of an application. It contains a list of menu topics in cascading buttons; each button is associated with a distinct pull-down menu containing commands that are grouped by common functionality. The use of a menu bar yields consistency across applications.

6-5: [Required]
The menu bar for the application contains only cascading buttons.

Note: When other buttons are included as topics in a menu bar, they inhibit menu browsing.

6-6: This item of the **OSF/Motif Style Guide** is not applicable. It is replaced by the following recommendation.

bn: [Recommended]
The standard menu bar entries are File, Edit, View, Options and Help. If the application provides that functionality to the user, it should be included in the menu bar under the appropriate name. The contents of these menu entries are discussed below in more detail.

Standard menu bar entries should be presented in the following order:

```
File Edit View Options Help
```

The application should exclude from its menu bar any item shown in the preceding text if the application does not support the associated function. For example, if the application does not support the ability to display its data in different views, then it should not include a View menu.

The application may add application-specific menus in between any of the standard menu items, with the following exceptions:

- The File menu, if present, is located in the first menu position on the left.
- The Help menu is located on the far right position.
- If File and Edit are present, they should be next to each other.

For example, the application may have:

```
File Edit <category1> <category2> View Options <category3> Help
```

bo: [Recommended]
Applications that are not file-oriented in nature (or that manage files transparently, not exposing this activity to the user) should replace the File menu with one or more application-specific menus.

Replacing the File menu:

Replacement1: `<app-label> Selected`

Replacement2: `<app-label><obj-type>`

Replacement3: `<obj-type>`

The application may use Replacement1 if the application has more than one object type. Items on <app-label> would be used for global actions that are not specific to an object type. The items in Selected are actions that pertain to objects that are currently selected, and may change depending on what objects are selected. If nothing is selected, this menu should have a single item that says (none selected). If an item is selected, but there are no items that apply to that object, this menu should have a single item that says (none).

The application may use Replacement2 if the application has a single object type. Actions that are global to the application are on <app-label>, and actions that are specific to the object type are on <obj-type>.

The application may use Replacement3 if the application has a single object type, and does not require an <app-label> menu. For example, a Print Manager might contain a Printer menu.

All other menu bar guidelines that apply to File-oriented applications also apply to non-File-oriented applications. Thus, the following menu bar would be valid:

```
<app-label> Selected Edit <category1> View <category2> Help
```

Applications that are complex or are extremely domain-specific (for example, an application for medical imaging and diagnosis of CAT scan data) may require other approaches to their menu bar design. For example,

```
<app-label><category1><category2> Selected Edit ...
<object-type> Options Help
```

bp: [Recommended]
Exit or Close should be located on the first (leftmost) menu of the application's menu bar.

20.7.1.4 File Menu Contents

The following requirements apply only in a left-to-right language environment in an English-language locale. The application must make the appropriate changes for other locales.

bq: [Required]
If the user chooses Exit or in any other manner indicates that the application should be terminated, but there are changes to the current file that have not been saved, the application displays a dialog box asking whether the changes should be saved before exiting.

Note: The user must always be given the opportunity to explicitly state whether unsaved changes should be saved or discarded. A dialog box similar to the one described should also be displayed if the user chooses Open from the File menu, but has not saved changes to the current file.

6-7: [Required]

If the application uses a File menu, it contains the following choices, with the specified functionality, when the actions are actually supported by the application (Section 6.2.1.5.1 of the **OSF/Motif Style Guide**).

New [Required]

This choice creates a new file. If the current client area will be used to display the new file, the application clears the existing data from the client area. If changes made to the current file will be lost, the application displays a dialog box, asking the user about saving changes. The mnemonic is N.

Open... [Required]

This choice opens an existing file by prompting the user for a file name with a dialog box. If changes made to the current file will be lost, the application displays a dialog box asking the user about saving changes. The is mnemonic is O.

Save [Required]

This choice saves the currently opened file without removing the existing contents of the client area. If the file has no name, the application displays a dialog box, prompting the user to enter a file name. The mnemonic is S.

Save As... [Required]

This choice saves the currently opened file under a new name by prompting the user for a file name with a dialog box. If the user tries to save the file using an existing name, the application displays a dialog box that warns the user about a possible loss of data. This choice does not remove the existing contents of the client area. The mnemonic is A.

Print [Recommended]

This choice schedules a file for printing. If the application needs specific information to print, it should display a dialog box requesting the information from the user. In this case, the menu entry is followed by an ellipsis (such as ''Print...''). The mnemonic is P.

Close [Recommended]

This choice closes the current primary window and its associated secondary windows. If the application uses only a single primary window or multiple dependent primary windows, this action is not supplied. The mnemonic is C.

Exit [Required]

This choice ends the current application and all windows associated with it. If changes made to the current file will be lost, the application displays a dialog box, asking the user about saving changes. The mnemonic is X.

Note: The use of a File menu with these common file operations yields consistency across applications.

<Object-Type> / Selected Menu Contents

br: [Recommended]

If the application uses an <object-type> menu or a Selected menu, it should contain the following choices, with the specified functionality, when the actions are actually supported by the application. Items should be presented to the user in the order listed below:

The <object-type> menu contains controls that allow the user to create instances of the object-type. Both the <object-type> and Selected menus allow the user to manipulate object instances. Additional items should be added to the <object-type> or Selected menus if they relate solely to the manipulation of objects managed by the application (as opposed to more generic services that the application might provide).

New... [Recommended]

This choice creates a new instance of the object-type. If appropriate, a dialog box should be presented allowing the user to specify the values for settings associated with that object.

Move To... [Optional]

This choice allows the user to move the selected objects into a folder. A file selection dialog box is displayed allowing the user to select the desired folder.

Copy To... [Optional]

This choice allows the user to copy the selected objects into a folder. A file selection dialog box is displayed allowing the user to select the desired folder.

Put in Workspace [Optional]

This choice allows the user to put a link for the object onto the XCDE desktop in the current workspace.

Any of the preceding three menu choices should be provided only if the objects managed by the application are able to reside as separate entities outside of the application's main window. For example, a printer object created by a printer management application might be able to be placed in a Folder window and function as an application unto itself. The application should also support drag-and-drop as a method for performing any of these actions.

Delete [Optional]

This choice removes the selected objects. A confirmation dialog box should be presented to the user before the object is actually deleted.

Properties [Recommended]

This choice displays a Properties window that should show the current values for settings associated with the selected object.

<Default Action> [Recommended]

This choice should enact the default action for the selected object. ''Open'' is a typical default.

Edit Menu Contents

The following requirements apply only in a left-to-right language environment in an English-language locale. The application must make the appropriate changes for other locales.

6-8: [Required]
If the application uses an Edit menu, it contains the following choices, with the specified functionality, when the actions are actually supported by the application (Section 6.2.1.5.2 of the **OSF/Motif Style Guide**):

Undo [Optional]
This choice reverses the most recently executed action. The mnemonic is U.

Cut [Optional]
This choice removes the selected portion of data from the client area and puts it on the clipboard. The mnemonic is T.

Copy [Optional]
This choice copies the selected portion of data from the client area and puts it on the clipboard. The mnemonic is C.

Copy Link [Optional]
This choice copies a link of the selected portion of data from the client area and puts it on the clipboard. The mnemonic is K.

Paste [Optional]
This choice pastes the contents of the clipboard into the client area. The mnemonic is P.

Paste Link [Optional]
This choice pastes a link of the data represented by the contents of the clipboard into the client area. The mnemonic is L.

Clear [Optional]
This choice removes a selected portion of data from the client area without copying it to the clipboard and does not compress the remaining data. The mnemonic is E.

Delete [Optional]
This choice removes a selected portion of data from the client area without copying it to the clipboard. The mnemonic is D.

Select All [Optional]
This choice sets the primary selection to be all the elements in a component of the client area.

Deselect All [Optional]
This choice removes from the primary selection all the elements in a component of the client area.

Select Pasted [Optional]
This choice sets the primary selection to the last element or elements pasted into a component of the client area.

Reselect [Optional]
This choice sets the primary selection to the last selected element or elements in a component of the client area. This action is available only in components that do not support persistent selections and only when the current selection is empty.

Promote [Optional]
This choice promotes to the primary selection the current selection of a component of the client area. This action is available only for components that support persistent selections.

Note: The use of an Edit menu with these common editing operations yields consistency across applications.

bs: [Recommended]
If the application does not provide an <object-type> or Selected menu, but allows the user to select data within the window and manage settings for the selected data, then it should provide a ''Properties ...'' choice as the last item in the Edit menu.

6-9: This item of the **OSF/Motif Style Guide** is not applicable.

View Menu

bt: [Recommended]
If the application provides a View menu, it should only contain functions that affect the way the current data is presented. It should not contain any option that alters the data itself.

Options Menu

bu: [Recommended]
If the application has global settings that control the way the application behaves, it should provide an Options menu from which these can be set.

20.7.1.5 Help Menu Contents

The following requirements apply only in a left-to-right language environment in an English-language locale. The application must make the appropriate changes for other locales.

bv: [Recommended]
If the application includes a Help menu, it should contain the following set of choices, with the specified functionality, when the actions are actually supported by the application. The Help choices included here supersede those listed for Motif 1.2.

Overview [Required]
This choice provides general information about the window from which help was accessed or about the application overall. The mnemonic is V. A separator is placed after this choice.

Index [Optional]
This choice provides an index listing topics for all help information available for the application. The mnemonic is I.

Table of Contents [Recommended]
This choice provides a table of contents listing topics for all help information available for the application. The mnemonic is C.

Tasks [Recommended]
This choice provides access to help information indicating how to perform different tasks using the application. The mnemonic is T.

Reference [Recommended]
This choice provides access to reference information. The mnemonic is R.

Tutorial [Optional]
This choice provides access to the application's tutorial. The mnemonic is L.

Keyboard [Optional]
This choice provides information about the application's use of function keys, mnemonics and keyboard accelerators. It also provides information on general XCDE use of such keys. The mnemonic is K.

Mouse [Optional]
This choice provides information about using a mouse with the application.The mnemonic is M.

Mouse and Keyboard [Optional]
This choice provides information about the application's use of function keys, mnemonics, keyboard accelerators and using a mouse with the application. It also provides information on general XCDE use of such keys. The mnemonic is M. This choice should be used instead of separate mouse and keyboard choices if this information is best presented together.

On Item [Recommended]
Initiates context-sensitive help by changing the shape of the pointer to the question mark pointer. When the user moves the pointer to a component and presses BSelect, any available context-sensitive help for the component is presented. The mnemonic is O. This choice is set off with separators on both sides.

Using Help [Required]
This choice provides information on how to use the XCDE Help Facility. The mnemonic is U. This choice is set off with separators on both sides.

About applicationname [Required]
Displays a dialog box indicating, minimally, the name and version of the application and displaying its icon or some other signature graphic for the application. The mnemonic is A.

6-10: This item of the **OSF/Motif Style Guide** is not applicable. It is replaced by item **bv**.

20.7.1.6 Attachment Menu Contents

bw: [Recommended]
If the application uses an attachment menu, it should contain the following choices, with the specified functionality, when the actions are actually supported by the application.

Add File... [Recommended]
This choice should select files and other items to be attached. A file selection box is displayed allowing the user to select the desired files to attach. The default button in the file selection box is Attach.

Save As... [Recommended]
This choice should save the currently selected attachments. The user is prompted with a file selection box for indicating where in the file system the attachments will be saved. When multiple attachments are selected, the name field is inactive and the current names of the attachments are used as the name of the new file. This menu item is active only when one or more attachments are selected.

Rename... [Recommended]
This choice renames the attachment icon. The application should provide in-line renaming of attachment icons such as File Manager uses. If the application cannot provide in-line renaming, then Rename allows the user to rename an attachment by displaying a dialog box, requesting the name from the user. This menu item is active only when a single attachment is selected. It is not active when multiple attachments are selected.

Delete [Recommended]
This choice deletes attachments from the attachment list. This menu item is active only when an attachment is selected.

Select All [Recommended]
This choice selects all the attachments in the attachment list.

20.7.1.7 Pop-up Menus

The following requirements apply only in a left-to-right language environment in an English-language locale. The application must make the appropriate changes for other locales.

bx: [Recommended]
If the application provides functions that apply to a data pane and not any specific element therein, then a pop-up menu should be provided that contains the frequently used data pane functions and is accessible by pressing BMenu when the mouse pointer is over the background of the pane or a non-selectable element within the pane.

by: [Recommended]
The application should provide a pop-up menu for any element that is selectable within its data pane.

Note: Pop-up menus provide access to frequently used functions and should be used pervasively throughout the XCDE desktop environment. A pop-up menu may contain a collection of options that appear in different menus available from the menu bar. For example, it may contain items from both the File and Edit menus.

bz: [Recommended]
When a pop-up menu is displayed over an unselected object, any action selected from the pop up applies to that object only and not to any other objects that might currently be selected.

Note: This helps to protect users from inadvertently applying an action to objects that they might not realise are currently selected. Pressing the menu button invokes a pop-up menu pertinent to the object under the mouse cursor whether it is selected or not or, if the object under the mouse cursor and other objects are selected, the pop up is pertinent to the selected set.

ca: [Recommended]
Every pop-up menu in the application should have a title that indicates the function the menu performs or the element on which it operates.

cb: [Recommended]
The functions accessible from within the application's pop-up menus should also be accessible from buttons displayed within the window or menus accessed through the menu bar.

Note: Because pop-up menus are hidden, they should only provide redundant access to functions available from more visible controls within the application's windows.

6-11: [Optional]
If the application uses any of the common pop-up menu actions, the actions function according to the following specifications (Section 6.2.1.6 of the **OSF/Motif Style Guide**). See item **cc** for supplemental guidelines.

Properties [Optional]
This choice displays a Properties dialog box that the user can use to set the properties of the component.

Undo [Optional]
This choice reverses the most recently executed action.

Primary Move [Optional]
This choice moves the contents of the primary selection to the component. This action is available only in editable components.

Primary Copy [Optional]
This choice copies the contents of the primary selection to the component. This action is available only in editable components.

Primary Link [Optional]
This choice places a link to the primary selection in the component. This action is available only in editable components.

Cut [Optional]
This choice cuts elements to the clipboard. If the menu is popped up in a selection, cuts the entire selection to the clipboard.

Copy [Optional]
This choice copies elements to the clipboard. If the menu is popped up in a selection, this action copies the entire selection to the clipboard.

Copy Link [Optional]
This choice copies a link of elements to the clipboard. If the menu is popped up in a selection, copies a link to the entire selection to the clipboard.

Paste [Optional]
This choice pastes the contents of the clipboard to the component. This action is available only in editable components.

Paste Link [Optional]
This choice pastes a link of the contents of the clipboard to the component. This action is available only in editable components.

Clear [Optional]
This choice removes a selected portion of data from the client area without copying it to the clipboard. If the menu is popped up in a selection, deletes the selection.

Delete [Optional]
This choice removes a selected portion of data from the client area without copying it to the clipboard. If the menu is popped up in a selection, deletes the selection.

Select All [Optional]
This choice sets the primary selection to be all of the elements in the collection with the pop-up menu.

Deselect All [Optional]
This choice deselects the current selection in the collection with the pop-up menu.

Select Pasted [Optional]
This choice sets the primary selection to be the last element or elements pasted into the collection with the pop-up menu.

Reselect [Optional]
This choice sets the primary selection to be the last selected element or elements in the component with the pop-up menu. This action is available only in components that do not support persistent selections and only when the current selection is empty.

Promote [Optional]
This choice promotes the current selection to the primary selection. It is available only in components that support persistent selections.

Note: The use of pop-up menus with these common actions yields consistency across applications.

cc: [Recommended]
Pop-up menus for selectable objects should contain the following set of choices, with the specified functionality, when the actions are actually supported by the application. These guidelines supplement item 6-11:.

Move To ... [Optional]
This choice allows the user to move the selected objects into a folder. A file selection dialog box is displayed allowing the user to select the desired folder.

Copy To ... [Optional]
This choice allows the user to copy the selected objects into a folder. A file selection dialog box is displayed allowing the user to select the desired folder.

Put on Workspace [Optional]
This choice allows the user to put a link for the selected objects onto the XCDE desktop in the current workspace.

Delete [Optional]
This choice deletes the selected object. A confirmation is displayed to the user before actually removing the object.

Properties ... [Recommended]
This choice displays a dialog box indicating the current settings for attributes associated with the selected object.

Help ... [Recommended]
This choice displays a help window pertaining to objects of the type selected.

cd: [Optional]
Choices within the pop-up menus are organised in the following manner:

```
<choices that manage the object, such as Open,
                                        Save or Properties>

----------- separator ----------------

<standard edit menu choices, such as Cut, Copy and Paste>

----------- separator ----------------

<other choices>
```

6-12: [Required]
When a pop-up menu is popped up in the context of a selection, any action that acts on elements acts on the entire selection (Section 6.2.1.6 of the **OSF/Motif Style Guide**).

Note: In the context of a selection, pop-up menu actions affect the entire selection.

20.7.1.8 Dialog Boxes

6-13: [Required]
Information dialog boxes do not interrupt the user's interaction with the application (Section 6.2.1.7.5 of the **OSF/Motif Style Guide**).

Note: An information dialog box conveys information to the user that does not require immediate attention, so it does not need to be modal.

20.7.1.9 Menu Design

ce: [Recommended]
If the selection of a menu item will result in the user being queried for more information, such as through the posting of a file selection dialog, the menu item should be followed by an ellipsis (''...''). This recommendation does not apply to menu items that will result in a simple warning or confirmation dialog being displayed.

Note: The use of an ellipsis helps set the user's expectation for the behavior of the interface. When users select an item without an ellipsis, they know that they can expect an immediate result.

cf: [Recommended]
Menus accessed from within the application should contain at least two menu items.

Note: No menu should contain only one item. If the application provides a menu with only one item, the application should move that item into another menu or make it a button within the window. The longer the menu, the more effort is needed for the user to access choices near the

bottom. If the application menu has a lot of choices, it should be broken up into two or more menus, or some items should be grouped into submenus.

cg: [Optional]
Submenus accessed from within the application contain at least three menu items.

Note: Submenus may be used to group like items into a single secondary cascading menu where putting the items into the primary cascading menu would make it too long. However, if the submenu contains only two options, the application should remove the secondary cascading menu and put the options into the primary cascading menu since it takes more effort for the user to access options located in a submenu.

ch: [Recommended]
No menu in the application should contain more than 15 choices.

Note: The longer the menu the more effort is needed for the user to access choices near the bottom. If the menu has a lot of choices, the application should break it up into two or more menus or group some items into submenus.

ci: [Optional]
If the application contains a menu that is expected to be accessed frequently, then a tear-off menu option is provided in that menu.

Note: The user should be able to tear-off frequently accessed menus so that these can remain posted on the desktop as the user uses the application.

cj: [Optional]
The application provides keyboard accelerators where appropriate. If specific menu items within a menu are expected to be used frequently, not the menu as a whole, then the application provides keyboard accelerators for these items and displays the keyboard accelerators in the associated menu to the right of the item to which they relate.

ck: [Recommended]
The labels used for items in the menu bar should not appear as options within the menus themselves.

Note: The names of items in the menu bar serve as titles for the options the menu contains. The name of the menu bar item should provide a term that accurately describes the concept of the category relating all of the menu items and should not be used as the name of any item within the menu itself.

cl: [Required]
Any menu choice that is not currently an appropriate selection is dimmed (insensitive).

Note: Dimmed controls cannot be activated by the user and should appear only when the inactive state is short-term (that is, there is something the user can do within the application or the desktop environment to make the control become active). When the control is persistently inactive (because of the current configuration of the application or system or a particular set of companion software is not currently installed), the control should be removed rather than dimmed.

cm: [Recommended]
If a menu item is used to indicate a selection state, the application should use a checkbox or radio button to indicate the state of the item. The application should use a checkbox if a single item is used to represent on or off states, and use radio buttons for multiple adjacent menu items in which only one of the items may be selected.

cn: [Required]
If radio buttons are used in a menu, the application uses separators between each set of radio buttons and other menu items.

co: [Recommended]
If a checkbox or radio button is used on a menu item, it should always be shown as either selected or not selected, and should not disappear when in the unselected state.

6-14: [Required]
If the application uses a tear-off button in a menu, the tear-off button is the first element in the menu (Section 6.2.3 of the **OSF/Motif Style Guide**).

Note: When a tear-off button is activated, the menu changes into a dialog box. The tear-off button must be the first item in the menu so that the entire contents of the menu are torn off.

6-15: [Required]
All menus must be wide enough to accommodate their widest elements (Section 6.2.3 of the **OSF/Motif Style Guide**).

Note: The ability to see the full label of each menu element allows the user to browse through a menu.

20.7.1.10 Dialog Box Design

The following requirements apply only in a left-to-right language environment in an English-language locale. The application must make the appropriate changes for other locales.

cp: [Recommended]
The title of dialog boxes used within the application should adhere to the conventions listed in the following table.

Dialog Box Title Conventions	
Window Usage	**Window Title Format**
Message	<app or object name> : <action or situation>
Progress	<app or object name> : <action> in Progress
Action (Command)	<app name> : <action>
Object Properties	<app name> : <object-type> Properties
Application Options	<app name> : <type> Options

cq: [Required]
Every dialog box in the application has at least one button that either performs the dialog box action and dismisses it or dismisses the dialog box without taking any action.

6-16: [Optional]
This item of the **OSF/Motif Style Guide** has been replaced by item **cr**.

cr: [Recommended]
If the application uses common dialog box actions, the actions should have the following specified functionality and labels:

Yes [Optional]
This label indicates an affirmative response to a question posed in the dialog box.

No [Optional]
This label indicates a negative response to a question posed in the dialog box.

OK [Optional]
This label applies any changes made to components in the dialog box and dismisses the dialog box.

<command> [Optional]
This label applies any changes made to components in the dialog box, performs the action associated with <command> and dismisses the dialog box.

This label should be used in lieu of OK, Yes or No as a button label when it provides more meaning to the user as to the action that will be performed when that button is clicked.

Apply [Optional]
This label applies any changes made to components in the dialog box and does not dismiss it.

Retry [Optional]
This label causes the task in progress to be attempted again.

Stop [Optional]
This label ends the task in progress at the next possible break point.

Pause [Optional]
This label causes the task in progress to pause.

Resume [Optional]
This label causes a task that has paused to resume.

Save As Defaults [Optional]
This label saves the current settings as the default settings that will appear the next time the window is displayed. The settings are not applied to any selected object and the dialog box is not dismissed.

A Save As Defaults button should be provided if it is expected that a user would want to use different default values for a set of controls within a dialog box than those that the application provide as the factory settings. For example, a Save As Defaults button might be provided in a ''New <object-type>'' window, allowing the user to indicate that whenever a new instance of that object-type is created, the current values should be displayed as the default settings instead of the values given by the application.

Reset [Optional]
This label cancels any changes that have not yet been applied by the application. The controls within the dialog box are reset to their state since the last time the dialog box action was applied. If no changes

have been applied within the current invocation of the dialog box, the controls are reset to the state when the dialog box was first displayed.

Reset to Factory [Optional]
This label cancels any changes that have not yet been applied. Components in the dialog box are reset to their default state and value as specified by the vendor that delivered the application (that is, the controls are restored to the original factory settings).

Cancel [Optional]
This label dismisses the dialog box without performing any actions not yet applied.

Help [Recommended]
This choice provides help for the dialog box.

cs: [Recommended]
Any visible control that is not currently active or whose setting is currently invalid should be dimmed.

Note: Dimmed controls cannot be activated by the user and should appear only when the inactive state is short-term (that is, there is something the user can do within the application or the desktop environment to make the control become active). When the control is persistently inactive (because of the current configuration of the application or system or a particular set of companion software is not currently installed), the control should be removed rather than dimmed.

ct: [Optional]
The application should keep the size of the dialog boxes to a minimum.

Note: On low-resolution displays, dialogs may take up most of the screen real estate and may run off the edge of the screen if not designed correctly.

cu: [Optional]
The application should avoid complexity in the dialog boxes. If the dialog box must support many functions, it should use an expandable dialog box (see Section 20.7.7 on page 420) or use more than one dialog in a nested fashion.

cv: [Optional]
The application should avoid the use of resize handles in the dialog box. However, the application can use resize handles when resizing is useful in allowing users to see more information; for example, when the dialog contains a scrolling list that is likely to be long and users will frequently need to search the list.

cw: [Optional]
Every dialog box in the application has exactly one default button that is activated when the Return key is pressed.

The default button should be associated with the most likely response from the user and should not be potentially destructive or irreversible. Some applications may have dialog boxes that do not reveal a default button until a specific set of fields has been filled out or otherwise manipulated.

cx: [Optional]
If a dialog box displayed by the application has controls that are considered to be advanced features, the application uses an expandable dialog box, or a multiple page dialog box, that provides a <category> option menu that allows the user to navigate to each page.

Controls that relate to advanced features should not be displayed with the set of options initially displayed to the user. The typical user should be presented with only those options that are necessary to use the basic functionality of the application. Users wanting to access advanced functionality within the dialog box can use the <category> option button. If there are few advanced controls or the settings for these controls are highly related to the settings of basic controls displayed in the dialog box (that is, the settings of the advanced controls change when the user changes settings for basic controls), the application might choose to provide an expandable dialog box (see Section 20.7.7 on page 420).

Property Windows

cy: [Required]
If the application provides settings that control the behaviour of the application, these settings are displayed in an application properties window that is accessible from an Options menu.

cz: [Recommended]
If the application manages objects and allows the user to see or modify settings for these objects, these settings should be displayed in an object properties window that is accessible from a ''Properties ...'' choice in the Edit, <object-type> or Selected menus, as well as from the pop-up menu associated with the object.

da: [Recommended]
If the application provides access to a Properties or Options window, this window should include the following set of buttons in the order listed, with the specified functionality, when supported by the application.

OK [Required]
This button applies any changes made to components in the dialog box and dismisses it. OK may be replaced by a more appropriate label; for example, Add. The alternative label should be a verb phrase.

Apply [Optional]
This button applies any changes made to components in the dialog box and does not dismiss it.

Reset [Required]
This button cancels any changes that have not yet been applied by the application. The controls within the dialog box are reset to their state since the last time the dialog box action was applied. If no changes have been applied within the current invocation of the dialog box, the controls are reset to their state as of when the dialog box was first displayed.

Reset to Factory [Optional]
This button cancels any changes that have not yet been applied. Components in the dialog box are reset to their default state or value as specified by the vendor that delivered the application (that is, the controls are restored to the original factory settings).

Cancel [Required]
This button dismisses the dialog box without performing any actions not yet applied.

Help [Required]
This button provides help for the dialog box.

db: [Recommended]
If the application provides a Properties window that displays settings for a selected object, the properties window should track the current selection and modify the state of any controls to reflect the properties of the currently selected object accurately.

20.7.1.11 File Selection Dialog Box

dc: [Optional]
If the application allows the user to open or save files, then it uses the standard XCDE file selection dialog box to allow the user to select specific files and directories.

Note: All user interactions with the file system should be facilitated by providing a point-and-click style of choosing files and directories. The user should never be forced to memorise and type in file paths. The user must be able to explore the contents and structure of the file system using scrolling lists. The expert user, however, should be able to directly enter a complete file path, as well as be able to use relative paths and environment variables such as *HOME*.

The labels and contents of the standard file selection dialog box may be modified as appropriate to make clear the particular context in which it is being used within the application.

dd: [Recommended]
If the application allows the objects it manages to exist as separate entities within folders or toolboxes within the desktop environment, a Copy To menu option or button should be provided that displays a file selection dialog box that allows the user to select the desired folder in which an icon for the object should be placed.

de: [Recommended]
The file selection dialog box should not display hidden (dot) directories or files, unless the user depends on using these types of files. If the application does support displaying hidden files, the application should supply a check box allowing users to toggle between showing and not showing hidden files, or else allow users to toggle between showing and hiding files at a global level in the application.

df: [Recommended]
The file selection dialog box should not show the full pathnames for files and directories, but should only show the relative names, except for the directory text field.

Note: The global XCDE setting should be:

```
XmFileSelectionBox.fullPathMode: false
```

Unless the application overrides this behaviour, the file selection dialog box should not show full pathnames in the list boxes.

dg: [Required]
In general, the file selection dialog box should recall the directory location that was previously set by the user.

Note: For example, if the user brings up Save As and navigates to **/users/jay/letters** to save the file, the next time the user brings up Save As, the file selection box should be in the directory **/users/jay/letters**. This information, however, should not be recalled once the user has closed the primary window; it should resort to the default directory.

20.7.1.12 About Dialog Box

dh: [Optional]
The About dialog box should contain a minimum set of information about the application that is visible in a single text pane. That minimum set should be:

- Application name
- Version number
- Release date
- Copyright

di: [Required]
The About dialog box contains a Close button. Other buttons, such as Help and More, are optional.

Other information contained in the About box might be:

dj: [Recommended]
Information about the operating system or other aspects required to run the application; for example, XCDE 1.0.

dk: [Optional]
A More Information dialog box for additional information such as development team credits, licencing, client or xhost information.

20.7.1.13 Dialog Box Layout

The following requirements apply only in a left-to-right language environment in an English-language locale. The application must make the appropriate changes for other locales.

dl: [Optional]
Controls within the dialog box are placed in a left-right, top-down layout based on the order in which the user is expected to fill out or choose options within the dialog box.

dm: [Required]
Push buttons that affect the dialog box as a whole, either by modifying its contents or layout, invoking the action of the dialog box or dismissing the dialog box, must be located at the bottom of the dialog box.

Note: In general, there should only be one row of buttons at the bottom of a dialog box. If the application has dialog boxes that contain several global buttons, it may be necessary to create two or more rows of buttons at the bottom of the dialog box. The last row should contain the standard dialog box buttons (OK, Reset, Cancel and Help). If a dialog box contains buttons that are not related to the dialog box as a whole, but relate to a specific

control within the dialog box, the buttons should be located with the control to which they relate.

dn: [Required]
If the application provides an Apply button within a dialog box, it also provides an OK button or command button that performs the dialog box action then dismisses it.

do: [Optional]
The application should not use cascading buttons within dialog boxes.

Note: In general, cascading buttons should only be used within menus and menu bars. The application should avoid their use in all other locations unless absolutely necessary.

dp: [Recommended]
If the application needs to use cascading buttons outside of a menu pane, it should use menu buttons.

20.7.1.14 Designing Drag and Drop

dq: [Recommended]
The application should provide a drag-and-drop (DND) method for all objects represented as icons and for all elements that the user can directly manipulate.

dr: [Recommended]
Any basic function that the application supports through drag and drop also should be supported through menus, buttons or dialog boxes.

Note: Drag and drop is considered an accelerator to functionality that is accessible through other user interface controls supported within the application. There should be no basic operation that is supported solely through drag and drop.

ds: [Recommended]
The application should use an icon graphic in a dialog box or window to indicate that objects within the dialog box or window can be dragged. The same icon graphic should be used to represent the draggable object in File Manager. The icon should be placed adjacent to any display of the contents of the object, if such display exists. If there is no such display, the icon should be placed in the upper right corner of the dialog box or window, unless a more suitable placement is determined. The icon should be 32×32 in size and have a label under it. The label should indicate what kind of object the icon graphic represents. The icon graphic should also be used as the source indicator in the drag icon.

dt: [Required]
During a drag operation, the application changes the current pointer to a drag icon.

Note: A drag icon provides visual feedback that a drag operation is in progress.

du: [Recommended]
During a drag operation, the application should change the current drag cursor to include a source indicator.

Note: A source indicator gives a visual representation of the elements being dragged.

dv: [Recommended]
During a drag operation, the application should change the current drag cursor to indicate invalid drop zones. It uses the standard XCDE Cannot pointer.

Note: The user must receive feedback as to where an object can and cannot be dropped. Minimally, feedback should be provided as to what are invalid drop zones. Feedback for valid drop zones should be enhanced by use of animation, recessing of the target drop zone and other such drag-over effects.

dw: [Recommended]
During a drag operation, the application should change the drop zone feedback to indicate a valid drop zone.

Note: Preferably, feedback for valid drop zones is enhanced by use of animation, recessing of the target drop zone and other such drag-over effects.

dx: [Required]
Pressing Cancel ends a drag-and-drop operation by canceling the drag in progress.

Note: Cancel provides a consistent way for the user to cancel a drag operation.

dy: [Required]
Releasing BTransfer (or BSelect) when not over a drop target ends a drag-and-drop operation.

Note: Releasing BTransfer (or BSelect) offers a consistent means of ending a drag operation.

dz: [Optional]
Any cursor change or drag-over effect the application uses occurs within 0.2 seconds of the mouse pointer reaching the target area and does not interfere, in any noticeable way, with the interactive performance of the drag operation.

ea: [Recommended]
In a collection that supports copy, move or link operations that can be performed by dragging, the feedback presented to the user during the drag operation should indicate whether a single object or multiple objects are being manipulated.

Note: Feedback provided during the drag operation should ensure that the user feels confident that the desired set of objects is being dragged. The drag icon used for multi-object drag operations should integrate the feedback used to indicate whether the operation is a move, copy or link.

eb: [Required]
After a successful transfer, the data is placed in the drop zone and any transfer icon used by the application is removed.

Note: A transfer icon can be used to represent the type of data being transferred during a drop operation. A successful drop operation results in the transfer of data.

ec: [Required]
If the application removes data upon the completion of a drag and drop, it does so only if the drag-and-drop transfer has completed successfully.

Note: If a drag-and-drop operation has been canceled or failed, the data or object that was the source of the drag must not be removed.

ed: [Required]
After a failed transfer, the data remains at the drag source and is not placed in the drop zone. Any transfer icon used by the application is removed.

Note: A failed drop operation does not result in the transfer of data.

ee: [Recommended]
If the user drops an object at an inappropriate drop zone within the application's window, the application should participate in the display of a snap back effect and also should post an error dialog box indicating the reason the drop was disallowed.

Note: The error message should state the context (for example, running action A on object B), what happened (for example, could not connect to system X) and how to correct the problem (for example, press the Help button to obtain information on diagnosing remote execution problems).

ef: [Recommended]
An application that accepts only single items should reject all multiple-item drops.

Note: There is no consistent method to determine which of the selected items the user really wants to drop.

eg: [Recommended]
If the application supports drag and drop as a means of loading a file into the application, the application should respond to this operation in a manner similar to when the file is loaded through more conventional means such as choosing Open from the File menu.

Note: As an accelerator, drag-and-drop loading of files should provide the same kind of feedback and behaviour as choosing Open from the File menu. For example, if changes to a currently loaded file have not yet been saved, the application should display a message dialog box asking whether the changes should first be saved before loading the new file.

6-17: [Required]
If the application provides any drag-and-drop help dialog boxes, they contain a Cancel button for canceling the drag-and-drop operation in progress (Section 6.2.5.4 of the **OSF/Motif Style Guide**).

Note: The Cancel button in the help dialog box provides a convenient way for the user to cancel a drag-and-drop operation.

20.7.2 Attachments

eh: [Recommended]
Drag and drop should not be the only method for attaching objects.

ei: [Recommended]
Double-clicking should be a shortcut for selecting the attachment and choosing the Open menu item for attachments and should never be the only way to access attachments.

ej: [Recommended]
When the user attempts to drop something into the attachment list that is not attachable, then the drop fails and the item should be snapped back to its source.

ek: [Recommended]
When the user has one or more attachments open for editing and attempts to do any operation that would result in potentially losing the user's edits, the user should be clearly warned and given the opportunity to save changes.

el: [Recommended]
When the user chooses something to attach from the file selection box that is not an attachable item, then the user should receive an error message explaining why the chosen item cannot be attached. For example: The folder ''My.Stuff'' cannot be attached because it is a folder.

Only documents, applications and scripts can be attached.

20.7.3 Installation

em: [Required]
An application should be installed to folders in the Application Manager, not directly to the front panel or subpanels. For consistency, only XCDE desktop components install to these locations. Users may choose to rearrange their front panel, but the application must not do this without user consent.

20.7.4 Interaction

6-18: [Required]
A warning dialog box allows the user to cancel the destructive action about which the dialog box is providing a warning (Section 6.3.2.2 of the **OSF/Motif Style Guide**).

Note: The user should have a way to cancel an operation that can cause destructive results.

en: [Required]
When the application displays a dialog box, it places the input focus at the first text field into which the user is allowed to type an entry or at the first control within the dialog box with which the user should interact.

Note: Input focus should always be placed at a predictable and intuitive location. The user should not be forced to set focus at the control most likely to be used when the window is displayed.

eo: [Recommended]
As the user presses the Tab key within dialog boxes of the application, the input focus should move to different controls within the window in a left-right, top-down order.

Note: These requirements apply only in a left-to-right language environment in an English-language locale. The application must make the appropriate changes for other locales.

ep: [Required]
There is always exactly one control within any window of the application that has the input focus if the window in which it resides has the input focus.

Note: If any window within the application has focus, some control within that window must have focus. The user should not have to explicitly set focus to a control within the window.

eq: [Optional]
When a text field within the application does not have the input focus, the text cursor is not displayed within that field.

Note: Although use of inactive text cursors is allowed within the Motif style, it is better to hide the text cursor on focus out rather than display the inactive text cursor. This makes it easier for the user to quickly scan the screen or a window and determine which text field currently has focus.

er: [Optional]
The application provides keyboard mnemonics for all buttons, menus and menu items displayed within the application.

Note: Once the user becomes adept at using the application, keyboard mnemonics provide the user a quick way to access functionality. Mnemonics also facilitate access to functionality from within keyboard-centric applications or windows. The user need not frequently switch between use of the mouse or use of the keyboard. Mnemonics should be provided pervasively throughout the user interface.

es: [Optional]
The application provides keyboard accelerators for those functions that are expected to be used frequently by the user.

Note: Keyboard accelerators provide the user who has become expert at using the application a quick way to access application functionality without having to go through menus and dialog boxes.

et: [Required]
Dialog boxes displayed by the application never block input to other applications within the desktop (that is, they are not system modal) unless it is absolutely essential that the user perform no other action in the desktop until the user responds to the dialog box.

Note: The application must allow the user the freedom to access information and tools within the user's desktop environment. An application should rarely block access to other applications and services within the environment.

eu: [Required]
Dialog boxes displayed by the application never block access to other functionality within the application (application modal) unless it is essential that the state of the application remains unchanged until the user responds to the dialog box.

ev: [Required]
If the application does not use the values of global environment settings (such as multiclick timeout intervals, drag thresholds, window colour settings, mouse left- or right-handedness), but instead uses its own values for these settings, then the application provides one or more Options dialog boxes that allow the user to change the values for these settings.

Note: In general, the application should not override the value of settings treated as global environment settings. These settings are controlled by the user through the XCDE Style Manager. If the application choose to ignore these settings and specify the own settings, then the application will behave inconsistently with other applications in the XCDE desktop. If the application nevertheless choose to provide the own values, then the application must provide the user a way to make the settings consistent

with the rest of the desktop.

20.7.5 Visuals

ew: [Recommended]
Any icons or graphics displayed by the application should be designed to be distinguishable on low- (640×480), medium- (800×600) and high- (mega-pixel) resolution displays. Alternatively, the application provides different sized visuals for low-, medium- and high-resolution displays.

Note: Desktop system configurations are including more high-resolution monitors. The user must be able to discern any visuals used by the application on these type of monitors. The embedded base, however, still contains many standard VGA monitors. The application's visuals must display well on these systems and should not appear overly large.

ex: [Recommended]
Any icons or graphics displayed by the application should be designed to display well on black-and-white and gray-scale monitors. These visuals also display well on low-colour (16) systems.

ey: [Recommended]
Icons should be used to represent only objects and applications.

Note: Icons provide a visual representation for objects and facilitate direct manipulation. If icons are used for other purposes (for example, as illustrations) where the user cannot drag them, select them and so on, it creates a confusing inconsistency.

ez: [Recommended]
Icons should use only the palette of 22 colours.

Note: The XCDE icon palette was chosen to maximise attractiveness and readability without using an unnecessary number of colours. Use of additional colours may cause undesirable colour shifting on the display.

fa: [Recommended]
Icons should be designed for international use.

Note: The application should not use text, symbols, humour, animals and other items that may be interpreted differently in other cultures.

fb: [Recommended]
Icons of sizes 16×16 and 32×32 should be left-aligned; any empty bits should be on the right side of the bounding box.

fc: [Recommended]
Icons of size 48×48 should be centered in the bounding box.

20.7.6 Toolbars

fd: [Required]
If the application uses a tool bar, it should be used only in windows with a menu bar.

fe: [Required]
Tool bars should contain only operations that are already available to the user in the application menus. All items in a tool bar should be redundant.

ff: [Required]
When an action represented by a tool bar icon is unavailable to the user, that icon should be made insensitive, with the associated stippled appearance. Whenever a menu item is made insensitive, the corresponding tool bar item is made insensitive as well.

fg: [Recommended]
The application should give users the option to hide the tool bar.

fh: [Required]
The tool bar container is placed directly under the menu bar and should be the same width as the window, as well as similar height to the menu bar.

fi: [Recommended]
If the application uses a tool bar, then the application should provide a status line in the same primary window as the tool bar.

Note: This status line should provide immediate feedback to the user as to the purpose of the button that the mouse is currently over or that has the keyboard focus. When the arrow is over a tool bar icon, the status line should display a brief definition of what the icon represents or what will happen when the user clicks the icon.

fj: [Recommended]
The application may provide labels under tool bar icons. These labels should serve to explain the purpose of the icon.

fk: [Recommended]
Drawn buttons in the tool bar should be the same width and height. Similar or related items should be grouped and groups should be evenly spaced across the tool bar.

fl: [Recommended]
All pixmaps in the tool bar should be the same size.

Note: This ensures that all the tool bar buttons are the same size.

fm: [Recommended]
The pixmap should be 24×24. The default for the drawn button is to resize itself according to the size of its label type, which, in this case, would be a pixmap.

20.7.7 Expandable Windows

fn: [Recommended]
The primary pane of the dialog box or window should contain all of the controls needed to complete the task. This should include all critical and frequently used functionality.

fo: [Recommended]
It is assumed that infrequently used features are placed in the secondary pane. The core functionality of the application should not depend on any controls placed in secondary panes.

fp: [Required]
Command buttons are aligned along the bottom of the dialog box. When the window is expanded to show a secondary pane, then buttons are moved to the bottom of the secondary pane. See Section 20.7 on page 393 for information about layout of action buttons in dialog boxes.

fq: [Recommended]
If important controls must be placed in the secondary pane, the application can specify that the window in question should be displayed in its expanded state by default.

fr: [Recommended]
The secondary pane should expand in the direction most consistent with users' expectations, the reading pattern of the language in which it will be displayed and the content of the information displayed.

fs: [Recommended]
If possible, the panes should have the same default width.

ft: [Required]
A separator is used to separate the primary pane from the secondary pane.

Note: The user must have clear visual feedback as to which elements are in the primary and which in the secondary panes of the expandable window.

fu: [Required]
If a window is resizable, any sizing changes are allocated to the pane containing scrolling lists or text fields whose displayed length is less than their stored length. If both panes contain scrollable controls, size changes are distributed evenly between the two panes. If neither pane contains scrollable controls, the window is not resizable.

fv: [Required]
The expandable window has one button that changes its label based on the state of the window.

fw: [Required]
The expand button has two labels that reflect the two states of the expandable window accurately. The current label indicates to the user what will happen if the user clicks the button.

Note: Examples of possible labels are Basic and Options, Expand and Contract, More and Less.

fx: [Optional]
The expand button may contain a graphic in addition to the label. This graphic should indicate the direction in which the window will expand or contract.

fy: [Recommended]
The button should appear in the lower left-hand corner of the window or dialog box for expansion in the vertical direction and in the lower right hand corner for expansion in the horizontal direction.

fz: [Required]
If the window or dialog box contains a scrolling list positioned to the far right side of the pane, the drawn button is not aligned with the scroll bar. For example, the button should be aligned with the list, not the scroll bar.

ga: [Required]
The application remembers the state of each window or dialog box (expanded or not expanded) independently (not collectively). The state must be changed only by the user and must be preserved until explicitly altered by the user.

gb: [Recommended]
The application should remember the state of each expandable window or dialog box across sessions, so that users do not have to manually configure the expandable windows each time the application is run.

Note: If appropriate, applications can provide a mechanism, as an option, to allow users to set the state of an expandable window globally for the application.

20.7.8 Messages

gc: [Recommended]
Messages displayed by the application should not assume that the user has any expert knowledge about computer systems in general or any operating system in particular.

Note: It is appropriate to assume that the user has knowledge about basic terms used within the desktop, such as files or programs. Such knowledge can be assumed to have been learned by the user through Tutorials, online help and user documentation. However, terminology that is typically understood only by an expert or frequent computer user should be avoided unless the application is specifically targeted at computer professionals. Likewise, messages returned to the application by the underlying operating system should not be passed through to the user, but instead, should be ''translated'' into language that can be understood by the novice user.

gd: [Recommended]
Error messages displayed by the application should indicate the possible cause of the error and indicate the possible actions the user can take in response.

ge: [Optional]
The application uses audio feedback, in addition to any messages displayed, to signal error conditions and events.

gf: [Optional]
The application should not rely on error messages from the underlying operating system and its library routines. Error messages from such routines are normally not seen by the user and even when the user does see them, they are usually too low-level and cryptic to be understood by non-programmers. The application should check for error conditions and use an Error dialog box to present an appropriate error message in terms of the user's actions and intentions.

gg: [Recommended]
The application should display a confirmation or warning message dialog box to the user when an action instigated by the user will be irreversible and potentially destructive with respect to the information stored within the system or the operation of the system or desktop environment.

gh: [Optional]
Urgent conditions that require immediate attention by the user, no matter which application or desktop service the user is currently accessing, are brought to the user's attention using audiovisual notification. The alarm is signaled in the current workspace regardless of the workspace in which the application resides.

Note: Some applications, such as network monitors or stock watch programs, may need to alert the user to some event. Both visual and audio alarms should be used to signal the user. The user should be able to acknowledge the alarm and cause it to cease.

gi: [Recommended]
The application should use footer messages only to communicate status, progress or information (help) messages. It should not use the footer to present error messages.

Note: The footer is a good location for prompt messages that help the user to determine how to choose options within a window or fill out a particular field. It should not be used to present error messages to the user or informational messages that are important for the user to notice. These should be presented in the appropriate style message dialog box.

gj: [Recommended]
The application should provide a Help button in all message dialog boxes, except those that contain self-explanatory messages.

Note: An application should be designed with both the expert and novice user in mind. The novice user must be able to access additional information clarifying the message, the circumstances under which it might have been displayed and what the user should do in response to the message.

gk: [Recommended]
The application should use the appropriate style dialog box for the display of messages to the user.

gl: [Optional]
An information dialog box is used to display status, completion of activity or other informative types of messages to which the user need not necessarily respond other than to acknowledge having read the message.

Note: Minimally, information dialog boxes should have an OK button so that the user can dismiss the dialog box. If there is additional information available about the situations under which the message is displayed or other references for the topic to which the message relates, then a Help button should be included.

gm: [Optional]

An error dialog box is used to display error messages to the user. The error dialog box displayed states what the error is and specifies why it occurred. The error dialog box contains a Help button so that the user may get additional information, unless the message is self-explanatory. The error dialog box contains an OK button that dismisses the dialog box.

Note: A Cancel button is not required for error dialog boxes unless the error resulted in the suspension of an activity that was in progress. In this case, the message should indicate whether the user has the option to continue the activity or stop it and the buttons for the dialog box should be Continue, Cancel and Help. In general, error dialog boxes should not be modal unless it is critical that the user not continue interacting with the application until the user has acknowledged having read the error message.

gn: [Optional]

A question dialog box is used to ask questions of the user. The question is clearly worded to indicate what a Yes response or a No response means. The buttons displayed are Yes, No and Help. Help provides additional information as to what the application will do in response to a Yes or No choice.

Note: Where possible, the application should extend the label for the Yes and No buttons to make it clear what action will be performed as a result of choosing either option. For example, if the user has made changes to a document and has not saved these but has chosen the application's Exit option, the application might display a question dialog box that asks ''Changes have not been saved. Do you want to save these before exiting?'' The buttons should be Save, Discard, Cancel and Help. These labels allow the more experienced user to click the correct button without having to read the question carefully and relate it to the button labels.

go: [Optional]

A warning dialog box is used to communicate the consequences of an action requested by the user that may result in the loss of data, system or application accessibility or some other undesirable event. The dialog box is presented before the action is performed and offers the user the opportunity to cancel the requested operation. The buttons displayed are Yes, No and Help or Continue, Cancel and Help. Help provides additional information on the consequences of performing the action requested.

Note: The use of Yes and No or Continue and Cancel depends on the wording of the message. The labels for Yes and No should be extended as suggested previously. Continue may be replaced with a label more specific to the action that will be performed.

gp: [Optional]

A working dialog box is used to display in-progress information to the user when this information is not displayed in the footer of the application's window. The dialog box contains a Stop button that allows the user to terminate the activity. The operation is terminated at the next appropriate breakpoint and a confirmation might be displayed asking whether the user really wants to stop the activity. The confirmation message might state the consequences of stopping the action.

gq: [Optional]
The application writes error messages to the XCDE error log when it is not appropriate to display these to the user in a message dialog box, but when the message may nevertheless be useful in diagnosing problems.

Note: The application might also write error messages that are displayed to the user in the error log if it would be valuable to the user or an administrator to refer to these messages at some later time. Messages written to the error log should provide additional information about the error and should state the context in which the error occurred.

gr: [Optional]
Informational messages should be left aligned and displayed in a light font in keeping with their unobtrusive nature. The margin where informational messages are displayed should not accept mouse focus.

gs: [Optional]
Progress messages should normally be displayed only while the operation is in progress. Notices and other information that is no longer valid should be removed within a few seconds to avoid confusion about whether or not the information is current.

20.7.9 Work-in-Progress Feedback

gt: [Recommended]
If any command chosen by the user is expected to take longer than two seconds to complete, but less than ten seconds, the application should display the standard busy pointer as feedback that the command is executing.

Note: The user must receive assurance that the application has ''heard'' the request and is working on it. If the results of the request cannot be displayed immediately, some feedback must be provided. The busy pointer should be displayed within 0.5 seconds of execution of the command.

gu: [Recommended]
If any command chosen by the user is expected to take longer than ten seconds to complete, the application should display a working dialog box or other feedback of similar character that indicates that the application is working on the request. The feedback should reveal progress toward completion of the activity.

Note: If an activity is expected to take a significant amount of time (ten seconds or more), the application should display feedback stronger than the busy pointer. Displaying the busy pointer for long amounts of time may lead the user to conclude that the application has become ''hung.'' A progress indicator should be displayed in these scenarios that indicates that the application is still functioning and is working on the user's request. The progress indicator should show how much of the activity has been completed and what amount remains.

gv: [Recommended]
When the application displays work-in-progress feedback to the user, it should not block access to other applications and services within the desktop environment.

Note: Multitasking should always be supported and, as such, the application should allow the user to access other services while it is busy performing some activity. Preferably, the user is also able to access other features within the application even though it is currently working on another request. When this is supported, the application should display an enhanced busy pointer that indicates that the application is busy but still willing to accept input.

20.8 Controls, Groups and Models

20.8.1 CheckButton

7-1: [Required]
The application uses check buttons to select settings that are not mutually exclusive. A check button graphically indicates its state with the presence or absence of a check mark (Chapter 9 of the **OSF/Motif Style Guide**).

Note: A check button is used to select settings that are not mutually exclusive. The user must know whether the button is set or not.

7-2: [Required]
When the user presses BSelect in a check button, the check button is armed. If the check button was previously unset, it is shown in the set state. If the check button was previously set, it is shown in the unset state (Chapter 9 of the **OSF/Motif Style Guide**).

Note: BSelect Press arms a check button and shows the result of activating it by releasing BSelect.

7-3: [Required]
When the user releases BSelect in the same check button in which the press occurred:

- If the check button was previously unset, it is set.
- If the check button was previously set, it is unset.

In all cases the check button is disarmed and, if the check button is in a menu, the menu is unposted (Chapter 9 of the **OSF/Motif Style Guide**).

BSelect Release activates a check button.

7-4: [Required]
When the user presses the Enter or Return key in a check button, if the check button is in a window with a default action, the default action is activated. If the check button is in a menu:

- If the check button was previously unset, it is set.
- If the check button was previously set, it is unset.
- In both cases, the check button is disarmed and the menu is unposted (Chapter 9 of the **OSF/Motif Style Guide**).

Note: The Enter and Return keys perform the default action of a window or activate a check button in a menu.

7-5: [Required]
When the user presses the Select key or Spacebar in a check button, if the check button was previously unset, it is set. If the check button was previously set, it is unset. In both cases, the check button is disarmed and, if the check button is in a menu, the menu is unposted (Chapter 9 of the **OSF/Motif Style Guide**).

Note: The Select key and Spacebar activate a check button.

20.8.2 ComboBox

gw: [Required]
In a list that can be scrolled, such as a scrollable list box, the application does not allow the cursor to wrap.

gx: [Required]
The application provides vertical scroll bars when some of the data is not visible in the combo box.

gy: [Recommended]
The application should provide horizontal scroll bars when elements are wider than the list box.

gz: [Recommended]
The application should display the elements in an order that is meaningful to the user.

ha: [Recommended]
The application should display an initial value from the list in the text-entry field. The application should display selected emphasis on the initial value so that typed text will replace the value.

hb: [Recommended]
The application should make the combination box large enough to display a minimum of six list items at a time.

hc: [Recommended]
When a user increases the size of the window in which the combo box is displayed, the application should increase the number of items displayed in the combo box.

hd: [Recommended]
When a user decreases the size of the window in which the combo box is displayed, the application should decrease the number of items displayed in the combo box. As a minimum, it should reduce the combo box to the text-entry field and a list box with one entry displayed. If the window is sized so that two list items cannot be displayed, it should clip the combo box.

20.8.3 CommandBox

7-6: [Required]
If the application uses a command box, it is composed of a text component with a command-line prompt for text input and a list component for a command history area. The list uses either the single selection or browse selection model (Chapter 9 of the **OSF/Motif Style Guide**).

Note: This requirement ensures the consistent appearance and operation of a command box across applications.

7-7: [Required]
When an element of a command box list is selected, its contents are placed in the text area (Chapter 9 of the **OSF/Motif Style Guide**).

Note: This requirement provides a convenient way of selecting a previously entered command.

7-8: [Required]
The list navigation actions Up Arrow, Down Arrow, Control+Begin and Control+End are available from the text component for moving the cursored element within the list and thus changing the contents of the text (Chapter 9 of the **OSF/Motif Style Guide**).

Note: These actions provide a convenient way to choose a command from the list while focus remains in the text component.

7-9: [Required]
The default action of the command box passes the command in the text area to the application for execution and adds the command to the end of the list (Chapter 9 of the **OSF/Motif Style Guide**).

Note: Maintaining a history of commands provides a convenient means of entering often-used commands.

20.8.4 File Selection Dialog Box

7-10: [Required]
If the application uses a file selection dialog box, it contains the following components (Chapter 9 of the **OSF/Motif Style Guide**):

- A directory text component showing the current directory path. The user can edit the directory text component and press Return or Enter to change the current directory.
- For applications that allow saving to different formats, an option button allowing users to specify the format when saving a file.
- A file name text component for displaying and editing a file name. This component is optional when the file selection box is used to choose an existing file or directory.
- A group of push buttons, including a command button, Update, Cancel and Help buttons. The command button is typically labeled Open or Save, but if there is another label that better describes the resulting action (such as Include), that label should be used. Activating the command button carries out the corresponding action and dismisses the file selection box.

he: [Recommended]
When the file selection box is used to specify an existing file (for example, to open a document), the command button is normally labeled Open and it should be the default action.

hf: [Recommended]
If the Update button is activated while a directory is selected in the contents list, the directory is opened, its contents are displayed in the contents list, and the directory text is updated.

hg: [Required]
If the Open button is activated while the appropriate file is selected in the contents list, the file is utilised by the application and the file selection box is dismissed.

hh: [Recommended]
When the file selection dialog box is used to choose an existing directory (for example, to install a set of files into the chosen directory) or to specify a new directory, the command button should be given an appropriate label, such as Install, Choose, Create or OK. If this button is activated while the appropriate directory is selected in the contents list, the directory is utilised by the application and the file selection box is dismissed.

hi: [Required]
When the file selection dialog box is used to choose an existing directory, there must also be an additional button, labeled Update, that is enabled whenever a directory is selected in the contents list, and opens the directory. This Update button is the default action.

hj: [Required]
When the file selection dialog box is used to specify a new file name (for example, a Save As dialog box), the command button is normally labeled Save and is the default action. This specification ensures the uniform appearance of a file selection box across applications.

hk: [Optional]
When the file selection dialog box is used to choose an existing file, files are shown in the contents list but they are all disabled. Double-clicking BSelect on a disabled file name has no effect.

hl: [Required]
The normal text navigation and editing functions are available in the text components for moving the cursor within each text component and changing the contents of the text.

Note: These actions provide a convenient way to choose a directory or file name from the corresponding list while focus remains in the text component.

7-11: This item of the **OSF/Motif Style Guide** is not applicable.

7-12: [Recommended]
Double-clicking BSelect on an item in the contents list should select that item and activate the default action. In all cases, double-clicking BSelect on a directory in the contents list should open that directory and display its contents in the contents list (the default action is Open).

- When the file selection box is used to choose an existing file, double-clicking BSelect on an appropriate file in the contents list should choose that file and dismiss the file selection box (the default action is Open).
- When the file selection box is used to choose an existing directory or to specify a new directory or file, files should be shown in the contents list, but they should all be disabled. Double-clicking BSelect on a disabled file name has no effect.

7-13: [Required]
The normal text navigation and editing functions are available in the text components for moving the cursor within each text component and changing the contents of the text.

7-14: This item of the **OSF/Motif Style Guide** is not applicable.

7-15: [Optional]
The application allows the user to select a file by scrolling through the list of file names and selecting the desired file or by entering the file name directly into the file selection text component. Selecting a file from the list causes that file name to appear in the file selection text area (Chapter 9 of the **OSF/Motif Style Guide**).

Note: This method of selecting a file should be consistent across applications.

7-16: [Required]
The application makes use of the selection when one of the following occurs (Chapter 9 of the **OSF/Motif Style Guide**):

- The user activates the command push button while an appropriate item is selected in the contents list.
- The user double-clicks BSelect on an appropriate file in the contents list.
- The user presses Return or Enter while the file name text component has the keyboard focus and contains an appropriate item.

7-17: [Required]
The file selection box displays the contents of a directory in the contents list when the file selection box is initialised, when the user presses Enter or Return in the directory text component and when the user opens a directory in the contents list. The contents list is updated each time the contents of the directory changes.

Note: This requirement ensures the consistent operation of a directory and file search in a file selection box.

hm: [Recommended]
If the user has opened the application without supplying a file name argument, the Open dialog box should use the user's home directory as the default directory.

Note: An exception to this recommenddation might be made if a clearly more useful directory can be identified; for example, the icon editor might default to **$HOME/.dt/icons**. An applications that allows editing should never default to a directory in which the user does not have read and write permission.

hn: [Required]
If the user has opened the application with a file name argument, the Open dialog box should default to the directory in which that file resides.

ho: [Optional]
When using the file selection dialog box in a Save As capacity, the application provides a default name of Untitled, places the location cursor in the file name field and highlights the file name text to create a ''delete pending type-in'' mode. If the current directory already has a file of that name, it creates a name Untitled2, and so forth.

hp: [Optional]
When using the file selection dialog box in a Save As capacity, the application adds a file name extension if the application supports file typing by extension and makes this extension visible in the file name field. It does not highlight the extension to create a ''delete pending type-in'' mode, but allows the user to modify the extension or delete it explicitly.

hq: [Optional]
The file selection box should display a directory that makes sense for the task. For example, when saving a new file from an editor, the file selection box should come up in the user's home directory. If the user navigates to some other directory within the file selection box, the application should remember that directory the next time it is brought up.

hr: [Optional]
Users should never be allowed to overwrite an existing file through the file selection box without a warning dialog box prompt.

hs: [Optional]
Keyboard focus should be placed in the file name field each time users bring up a file selection dialog box.

ht: [Optional]
Directory and file name lists should be presented alphabetically, case insensitive. The first item on the directory list should be the parent directory and it should be labeled ''..''.

hu: [Optional]
Labels should be clear. In the English language, the application should use the following labels for the file selection dialog box fields and lists:

File Selection Dialog Box Labels	
Component	**Label**
Directory text field	`Enter Path or Folder Name:`
Filter text Field	`Filter:`
Directory list	`Folders:`
Contents list	`Files:`
File text field	`Enter File Name:*`

hv: [Optional]
Application developers can make this label more instructive and specific, such as ''Enter File to Open'' for Open dialog boxes, and so forth.

Note: These labels should be the default labels. If they are not set by default, the application must set them through resources in the application's app-defaults file.

20.8.5 List

7-18: [Required]
Within a list component, the application uses the Up Arrow key to move the location cursor to the previous item in the list and the Down Arrow key to move the location cursor to the next item in the list. In a scrollable list, the Left Arrow key scrolls the list one character to the left and the Right Arrow key scrolls the list one character to the right (Chapter 9 of the **OSF/Motif Style Guide**).

Note: The arrow keys provide a consistent means of moving the location cursor within a list component.

7-19: [Required]
Within a list component, the application uses Control+Begin to move the location cursor to the first item in the list and Control+End to move the location cursor to the

last item in the list. In a scrollable list, the Begin key moves the horizontal scroll region so that the leftmost edge of the list is visible and the End key moves the horizontal scroll region so that the rightmost edge of the list is visible (Chapter 9 of the **OSF/Motif Style Guide**).

Note: These keys offer a convenient mechanism for moving the location cursor quickly through a list.

7-20: [Required]
Within a scrollable list, the Page Up key moves the location cursor to the item one page up in the list and the Page Down key moves the location cursor to the item one page down in the list. In a scrollable list, the Page Left key (or Control+Page Up) scrolls the list one page to the left and the Page Right key (or Control+Page Down) scrolls the list one page to the right (Chapter 9 of the **OSF/Motif Style Guide**).

Note: These keys offer a convenient mechanism for paging through a list.

7-21: [Required]
Within a list component, the application uses BSelect Click 2 to select the item that was double-clicked and then initiate any default action for the window (Chapter 9 of the **OSF/Motif Style Guide**).

Note: Double-clicking using BSelect provides a consistent way of activating the default action for a list.

20.8.6 Option Button

7-22: [Required]
If the application uses option buttons, the label for the button is the last selection made from the option button (Chapter 9 of the **OSF/Motif Style Guide**).

Note: An option button is used to post an option menu which allows the user to select from a number of choices. The label of an option button must display the most recent selection from the associated option menu.

7-23: [Required]
When the user presses BSelect or BMenu in an option button, the associated option menu is posted (Chapter 9 of the **OSF/Motif Style Guide**).

Note: BSelect Press is a consistent way of activating an option button.

7-24: [Required]
When the user releases BSelect or BMenu within the same option button that the press occurred in, the associated option menu is posted if it was not posted at the time of the press. When the user releases BSelect or BMenu outside of the option button, the associated option menu is unposted (Chapter 9 of the **OSF/Motif Style Guide**).

Note: BSelect Release or BMenu Release posts or unposts an option menu, depending on whether the release occurs inside the option button and whether the option menu was posted at the time of the press.

7-25: [Required]
When the user presses the Select key or Spacebar in an option button, the associated option menu is posted (Chapter 9 of the **OSF/Motif Style Guide**).

Note: The Select key or Spacebar posts an option menu from the keyboard.

20.8.7 Paned Window

7-26: [Required]
If the application uses paned windows, they are composed of any number of groups of components, called panes, each separated by a sash and a separator. The panes, sashes and separators are grouped linearly, either horizontally or vertically. A sash is the handle on a separator between two panes that is used to adjust the position of the separator (Chapter 9 of the **OSF/Motif Style Guide**).

Note: This requirement ensures the consistent appearance of a paned window across applications.

20.8.8 Panel

7-27: [Required]
The Down Arrow, Left Arrow, Right Arrow and Up Arrow directional keys navigate among components in a Panel (Chapter 9 of the **OSF/Motif Style Guide**).

Note: A panel group organises a collection of basic controls in a horizontal, vertical or two-dimensional layout. The directional keys are used to navigate among the controls.

20.8.9 Push Button

7-28: [Required]
When the user presses BSelect in a push button, the push button is armed. When the user releases BSelect in the same push button that the press occurred in, the push button is disarmed and activated. When the user releases BSelect outside the push button, the push button is disarmed but not activated (Chapter 9 of the **OSF/Motif Style Guide**).

Note: BSelect provides a consistent means of activating a push button.

7-29: [Required]
When the user presses the Enter or Return key in a push button that is in a window with a default action, the push button is activated. When the user presses the Enter or Return key in a push button in a menu, the push button is activated and the menu is unposted (Chapter 9 of the **OSF/Motif Style Guide**).

Note: The Enter and Return keys activate a dialog box or a push button in a menu.

7-30: [Required]
When the user presses the Select key or Spacebar in a push button, the push button is activated. If the push button is in a menu, the menu is unposted (Chapter 9 of the **OSF/Motif Style Guide**).

Note: The Select key and Spacebar activate a push button.

20.8.10 Radio Button

7-31: [Required]
If the application uses radio buttons, each button graphically indicates its state (Chapter 9 of the **OSF/Motif Style Guide**).

Note: Radio buttons are used to represent a panel of mutually exclusive selections. The user must know which button in the panel is set.

7-32: [Required]
When the user presses BSelect in a radio button, the radio button is armed. If the radio button was previously unset, it is shown in the set state (Chapter 9 of the **OSF/Motif Style Guide**).

Note: BSelect Press arms a radio button and shows the result of activating it by releasing BSelect.

7-33: [Required]
When the user releases BSelect in the same radio button that the press occurred in and the radio button was previously unset, it is set and any other radio button in the same panel that was previously set is unset. The radio button is disarmed and, if the radio button is in a menu, the menu is unposted (Chapter 9 of the **OSF/Motif Style Guide**).

Note: BSelect Release activates a radio button.

7-34: [Required]
When the user presses the Enter or Return key in a radio button, if the radio button is in a window with a default action, the default action is activated. If the radio button is in a menu (Chapter 9 of the **OSF/Motif Style Guide**):

- If the radio button was previously unset, it is set and any other radio button in the same panel that was previously set is unset.
- The radio button is disarmed and the menu is unposted.

Note: The Enter and Return keys perform the default action of a window or activate a radio button in a menu.

7-35: [Required]
When the user presses the Select key or Spacebar in a radio button, if the radio button was previously unset, it is set and any other radio button in the same panel that was previously set is unset. The radio button is disarmed and, if the radio button is in a menu, the menu is unposted (Chapter 9 of the **OSF/Motif Style Guide**).

Note: The Select key and Spacebar activate a radio button.

20.8.11 Sash

7-36: [Required]
Within a paned window, the application uses a sash to adjust the position of a separator, which adjusts the sizes of the panes next to it. As a sash is moved, the pane in the direction of the sash movement gets smaller and the opposite pane gets larger by an equal amount (Chapter 9 of the **OSF/Motif Style Guide**).

Note: This requirement results in the uniform operation of a paned window across applications.

7-37: [Required]
Within a sash, BSelect Motion or BTransfer Motion causes the sash to track the movement of the pointer. In a vertically oriented paned window, the sash tracks the vertical position of the pointer. In a horizontally oriented paned window, the pane tracks the horizontal position of the pointer (Chapter 9 of the **OSF/Motif Style Guide**).

Note: BSelect, mouse button 1 and BTransfer, mouse button 2, provide a consistent means of moving a sash in a paned window using the mouse.

7-38: [Required]
The Up Arrow and Down Arrow keys (for a sash that can move vertically) and the Left Arrow and Right Arrow keys (for a sash that can move horizontally) move the sash one increment in the specified direction (Chapter 9 of the **OSF/Motif Style Guide**).

Note: The arrow keys offer a uniform means of moving a sash in a paned window.

7-39: [Required]
Control+Up Arrow and Control+Down Arrow (for a sash that can move vertically) and Control+Left Arrow and Control+Right Arrow (for a sash that can move horizontally) move the sash one large increment in the specified direction (Chapter 9 of the **OSF/Motif Style Guide**).

Note: These keys provide a convenient way of moving a sash quickly in a paned window.

20.8.12 Scale

7-40: [Required]
If a scale has arrow buttons, the application uses BSelect Press in an arrow button to move the slider one increment in the direction of the side of the slider on which the button was pressed and autorepeats until the button is released (Chapter 9 of the **OSF/Motif Style Guide**).

Note: BSelect Press provides a consistent means of adjusting a scale component using the mouse.

7-41: [Required]
In a scale trough, if the scale has tick marks, BSelect Press moves the slider one major tick mark in the direction of the side of the slider on which the trough was pressed and autorepeats until the button is released. If the scale does not have tick marks, BSelect Press in the trough moves the slider one large increment in the direction of the side of the slider on which the trough was pressed and autorepeats until the button is released (Chapter 9 of the **OSF/Motif Style Guide**).

Note: BSelect Press provides a consistent means of adjusting a scale component using the mouse.

7-42: [Required]
Within a scale slider, BSelect Motion causes the slider to track the position of the pointer. In a vertical scale, the slider tracks the vertical position of the pointer. In a horizontal scale, the slider tracks the horizontal position of the pointer (Chapter 9 of the **OSF/Motif Style Guide**).

Note: BSelect Motion offers a convenient way to adjust a scale component precisely using the mouse.

7-43: [Required]
Within a scale slider or trough, BTransfer Motion positions the slider to the point of the button press and then causes the slider to track the position of the pointer. In a vertical scale, the slider tracks the vertical position of the pointer. In a horizontal scale, the slider tracks the horizontal position of the pointer (Chapter 9 of the **OSF/Motif Style Guide**).

Note: BTransfer Motion provides another convenient way to adjust a scale component precisely using the mouse.

7-44: [Required]
If a mouse-based sliding action is in progress, the Cancel key cancels the sliding action and returns the slider to its position prior to the start of the sliding operation (Chapter 9 of the **OSF/Motif Style Guide**).

Note: The Cancel key provides a consistent way for the user to cancel a mouse-based sliding action.

7-45: [Required]
In a vertical scale, the Up Arrow and Down Arrow keys move the slider one increment in the specified direction. In a horizontal scale, the Left Arrow and Right Arrow keys move the slider one increment in the specified direction (Chapter 9 of the **OSF/Motif Style Guide**).

Note: The arrow keys provide a uniform way of adjusting the slider in a scale component using the keyboard.

7-46: [Required]
In a vertical scale, Control+Up Arrow and Control+Down Arrow move the slider one large increment in the specified direction. In a horizontal scale, Control+Left Arrow and Control+Right Arrow move the slider one large increment in the specified direction (Chapter 9 of the **OSF/Motif Style Guide**).

Note: These keys provide a convenient way of adjusting the slider in a scale component quickly using the keyboard.

7-47: [Required]
The application uses the Begin key or Control+Begin to move the slider to its minimum value. The End key or Control+End moves the slider to its maximum value (Chapter 9 of the **OSF/Motif Style Guide**).

Note: These keys provide a convenient mechanism for setting a scale to its minimum or maximum value using the keyboard.

20.8.13 ScrollBar

7-48: [Required]
Within a scroll bar, the application uses BSelect Press in an arrow button to move the slider one increment in the direction of the side of the slider on which the button was pressed and autorepeats until the button is released (Chapter 9 of the **OSF/Motif Style Guide**).

Note: BSelect Press provides a consistent means of adjusting a scroll bar using the mouse.

7-49: [Required]
In the trough of a scroll bar, BSelect Press moves the slider one page in the direction of the side of the slider on which the trough was pressed and autorepeats until the button is released (Chapter 9 of the **OSF/Motif Style Guide**).

Note: BSelect Press provides a consistent means of adjusting a scroll bar using the mouse.

7-50: [Required]
Within a scroll-bar slider, BSelect Motion causes the slider to track the position of the pointer. In a vertical scroll bar, the slider tracks the vertical position of the pointer. In a horizontal scroll bar, the slider tracks the horizontal position of the pointer (Chapter 9 of the **OSF/Motif Style Guide**).

Note: BSelect Motion offers a convenient way to adjust a scroll bar precisely using the mouse.

7-51: [Required]
Within a scroll-bar slider or trough, BTransfer Motion positions the slider to the point of the button press and then causes the slider to track the position of the pointer. In a vertical scroll bar, the slider tracks the vertical position of the pointer. In a horizontal scroll bar, the slider tracks the horizontal position of the pointer (Chapter 9 of the **OSF/Motif Style Guide**).

Note: BTransfer Motion offers another convenient way to adjust a scroll bar precisely using the mouse.

7-52: [Required]
If a mouse-based scrolling action is in progress, pressing the Cancel key cancels the scrolling action and returns the slider to its position prior to the start of the scrolling operation (Chapter 9 of the **OSF/Motif Style Guide**).

Note: The Cancel key provides a consistent way for the user to cancel a mouse-based scrolling action.

7-53: [Required]
In a vertical scroll bar, the Up Arrow and Down Arrow keys move the slider one increment in the specified direction. In a horizontal scroll bar, the Left Arrow and Right Arrow keys move the slider one increment in the specified direction (Chapter 9 of the **OSF/Motif Style Guide**).

Note: The arrow keys provide a uniform means of adjusting a scroll bar using the keyboard.

7-54: [Required]
In a vertical scroll bar, Control+Up Arrow and Control+Down Arrow move the slider one large increment in the specified direction. Control+Left Arrow and Control+Right Arrow move the slider one large increment in the specified direction (Chapter 9 of the **OSF/Motif Style Guide**).

Note: These keys provide a convenient way of adjusting a scroll bar quickly using the keyboard.

7-55: [Required]
The application uses the Page Up and Page Down keys to move the slider in a vertical scroll bar one page in the specified direction. The Page Left key (or Control+Page Up) and the Page Right key (or Control+Page Down) move the slider in a horizontal scroll bar one page in the specified direction (Chapter 9 of the **OSF/Motif Style Guide**).

Note: These keys allow for the convenient movement of the slider in a scroll bar using the keyboard.

7-56: [Required]
The application uses the Begin key or Control+Begin to move the slider to the minimum value. The End key or Control+End moves the slider to the maximum value (Chapter 9).

Note: These keys offer a convenient mechanism for setting a scroll bar to its minimum or maximum value using the keyboard.

20.8.14 SelectionBox

7-57: [Required]
If the application uses a selection box, it is composed of at least a text component for the selected alternative and a list component above the text component for presenting alternatives. The list uses either the single selection or browse selection model. Selecting an element from the list places the selected element in the text component (Chapter 9 of the **OSF/Motif Style Guide**).

Note: This requirement ensures the consistent appearance and operation of a selection box across applications.

7-58: [Required]
The list navigation actions Up Arrow, Down Arrow, Control+Begin and Control+End are available from the text component for moving the cursored element within the list and thus changing the contents of the text (Chapter 9 of the **OSF/Motif Style Guide**).

Note: These actions provide a convenient way to choose an element from the list while focus remains in the text component.

20.8.15 Spin Box

hw: [Required]
The application presents the items as a ring of items that wrap. For example, if a user is at the largest number and presses the up arrow, the smallest number is displayed and vice versa so that the user can spin through all the items by pressing the same arrow.

hx: [Required]
The application moves through the items in a spin box as shown in the following table.

Spin Box Movements		
Movement	**Keys**	**Example**
Toward the beginning of the list	left arrow, down arrow	Chronological: If Tuesday is displayed, move to Monday when the user presses the left or down arrow. Magnitude: If 15 is displayed, move to 14 when the user presses the left or down arrow.
Toward the end of the list	right arrow, up arrow	Chronological: If Tuesday is displayed, move to Wednesday when the user presses the right or up arrow. Magnitude: If 15 is displayed, move to 16 when the user presses the right or up arrow.

hy: [Recommended]
Values can be set using the arrow buttons or through keyboard input. Values should be evaluated immediately upon entry. If a value is entered that is already in the list, scroll to the position of that entry in the list.

hz: [Recommended]
If entry of non-listed items is permitted, the application should use the following behavior. When a new value is entered, it should scroll the list to the position appropriate for the new entry. If the user scrolls off the new entry, it should scroll to the next appropriate value in the list and the keyboard-entered value is lost.

ia: [Recommended]
On entry of an invalid value, an auditory warning and error message should be provided.

20.8.16 Text

7-59: [Required]
In a multi-line text component, the Up Arrow key moves the location cursor up one line and the Down Arrow key moves the location cursor down one line. In a single-line text component, the Up Arrow key navigates upward to the previous component and the Down Arrow key navigates downward to the next component, if the text component is designed to act like a basic control (Chapter 9 of the **OSF/Motif Style Guide**).

Note: The up and down arrow keys provide a uniform means of navigation within text components.

7-60: [Required]
The Left Arrow key moves the location cursor left one character and the Right Arrow key moves the location cursor right one character (Chapter 9 of the **OSF/Motif Style Guide**).

Note: The left and right arrow keys offer a consistent way of navigating within text components.

7-61: [Required]
In a text component used generally to hold multiple words, Control+Right Arrow moves the location cursor to the right by a word and Control+Left Arrow moves the location cursor to the left by a word (Chapter 9 of the **OSF/Motif Style Guide**).

Note: Control+Right Arrow and Control+Left Arrow provide a uniform way of navigating by words in a text component. Moving right by a word means that the location cursor is placed before the first character that is not a space, tab or newline character after the next space, tab or newline. Moving left by a word means that the location cursor is placed after the first space, tab or newline character preceding the first previous character that is not a space, tab or newline.

7-62: [Required]
In a text component used generally to hold multiple words, the Begin key moves the location cursor to the beginning of the line and the End key moves the location cursor to the end of the line (Chapter 9 of the **OSF/Motif Style Guide**).

Note: These keys allow the user to move quickly to the beginning or end of a line of text in a text component.

7-63: [Required]
In a multi-line text component, Control+Begin moves the location cursor to the beginning of the file and Control+End moves the location cursor to the end of the file (Chapter 9 of the **OSF/Motif Style Guide**).

Note: These keys permit the user to move quickly to the beginning or end of a file in a text component.

7-64: [Required]
The application uses Spacebar or Shift+Spacebar to insert a space in a text component. Modifying these with Control invokes the normal selection function (Chapter 9 of the **OSF/Motif Style Guide**).

Note: This requirement ensures that selection is available from the keyboard in a text component.

7-65: [Required]
Return in a multi-line text component inserts a carriage return. The Enter key or Control+Return invokes the default action (Chapter 9 of the **OSF/Motif Style Guide**).

Note: This requirement ensures that activation is available from the keyboard in a text component.

7-66: [Required]
In a multi-line text component, Tab is used for tabbing. In a single-line text component, Tab is used either for tabbing or to move to the next field (Chapter 9 of the **OSF/Motif Style Guide**).

7-67: [Required]
If a text component supports replace mode, Insert toggles between insert mode and replace mode.

By default, the component starts in insert mode, where the location cursor is between two characters. In insert mode, typing a character inserts the character at the position of the location cursor.

In replace mode, the location cursor is on a character. Typing a character replaces the current character with that newly entered character and moves the location cursor to the next character, selecting it (Chapter 9 of the **OSF/Motif Style Guide**).

Note: These requirements ensure the uniform operation of a text component with a replace mode.

7-68: [Required]
The application uses BSelect Click 2 to select text a word at a time (Chapter 9 of the **OSF/Motif Style Guide**).

Note: Double-clicking with mouse button 1 provides a convenient mechanism for selecting words in a text component.

20.8.17 Gauge

ib: [Required]
A gauge is similar to a scale except that a gauge is a display-only device with no user interactions. The appearance of a gauge is similar to a scale, but the gauge lacks a scale slider.

ic: [Optional]
Despite being a display-only device, a gauge should get keyboard focus so that the user can access Help or Settings for that control.

20.9 Accessibility

id: [Recommended]
All application functions should be accessible from the keyboard.

ie: [Recommended]
Colours should not be hard coded (in other words, they should not be compiled into the program and made unchangeable by the user).

if: [Recommended]
Graphic attributes, such as line, border and shadow, should not be hard coded.

ig: [Recommended]
Font sizes and styles should not be hard coded.

ih: [Recommended]
The application should use descriptive names for widgets.

Note: Such descriptive names for widgets using graphics instead of text (for example, palette items and icons) allow screen reading software to provide descriptive information to blind users.

ii: [Recommended]
Interactions should not depend upon the assumption that a user will hear an audible notification.

ij: [Recommended]
Users should be be able to choose to receive cues as audio or visual information, where appropriate.

ik: [Recommended]
The application should not overuse or rely exclusively on audible information.

il: [Recommended]
Users should be able to choose to configure the frequency and volume of audible cues.

im: [Recommended]
Tear-off menus and user configurable menus for key application features should be provided for users with language and cognitive disabilities.

in: [Recommended]

Application key mappings should not conflict with existing system level key mappings reserved for access features in the X Windows server as shown in the following table.

Keyboard Mappings for Server-Level Access Features	
Keyboard Mapping	**Reserved For**
Five consecutive clicks of Shift key	On / Off for StickyKeys
Shift key held down 8 seconds	On / Off for SlowKeys and RepeatKeys
Six consecutive clicks of Control key	On / Off for screen reader numeric keypad functions
Six consecutive clicks of Alt key	Reserved for future access use

Index