Solaris 共通デスクトップ環境 プログラマーズ・ガイド

Sun Microsystems, Inc.
901 San Antonio Road
Palo Alto, CA 94303
U.S.A. 650–960–1300

Part Number 806-2737-10
2000 年 3 月

原典: *Solaris Common Desktop Environment: Programmer's Guide*

Part No: 806-1364-10

Revision A

Adobe PostScript

目次

はじめに

このマニュアルは、共通デスクトップ環境 (Common Desktop Environment、CDE) 開発環境を説明します。Motif、X、UNIX、または C プログラミングの知識があることを前提としています。

対象読者

このマニュアルは、既存のアプリケーションの CDE への統合、または CDE の機能を使用する新しいアプリケーションの開発に関心があるプログラマを対象としています。

このマニュアルを読む前に

このマニュアルは、プログラミング情報を集めたものです。CDE へのアプリケーションの統合を開始する前に、13ページの「関連マニュアル」にリストされているマニュアルをお読みください。

『共通デスクトップ環境 プログラマ概要』は、CDE の説明と、プログラミング環境を紹介しています。

内容の紹介

このマニュアルは、3 つのパートから構成にされています。各部に、共通デスクトップ環境の各要素の詳しい説明、概念図、各要素の具体的な使い方の説明、コーディング例があります。

パート I

アプリケーション・レベルと印刷レベルの登録方法を説明します。

第 1 章

既存のアプリケーションの CDE への基本的な統合に必要な手順を説明します。

パート II

既存のアプリケーションを共通デスクトップ環境に統合する方法を説明します。

第 2 章

一般的な標準フォントの記述を使用して、CDE 準拠システム上でアプリケーションに最も近いフォントを使用する方法を説明します。

第 3 章

情報とエラー・メッセージを表示するための一般的なモデルを説明します。

第 4 章

ICCM セッション管理プロトコルを説明し、セッション・マネージャとのアプリケーションの統合の例を示します。

第 5 章

ドラッグ & ドロップのユーザ・モデル、新しいドラッグ & ドロップのアプリケーション・プログラム・インタフェース (API)、およびドラッグ & ドロップの使い方を説明します。

パート III

新しいアプリケーションをセッション・マネージャおよびドラッグ & ドロップと統合する方法を説明します。また、ロケールがログイン・マネージャ、ウィンドウ・マネージャ、および端末エミュレータに与える影響についても説明します。

第 6 章

アプリケーションを特殊な方法でワークスペース・マネージャと統合する方法を説明します。

第 7 章

CDE の一部として提供されるカスタム・ウィジェットの使い方を説明します。

第 8 章

アプリケーションの中でアクションを作成する方法を説明します。

第 9 章

データ型関数とデータ型データベースの使い方を説明します。

第 10 章

カレンダ API について、関数、データ構造、カレンダ属性、およびエントリ属性などを説明します。カレンダ API の使い方も説明します。

用語集

このマニュアルで使われている語句とその定義のリストです。

関連マニュアル

CDE へのアプリケーションの統合を開始する前に、他のマニュアルも参照してください。マニュアルのリストについては、次の 14ページの「開発環境用マニュアル」の節も参照してください。

実行環境用のマニュアルは、次のとおりです。

- 『*Solaris* 共通デスクトップ環境 ユーザーズ・ガイド』
- 『*Solaris* 共通デスクトップ環境への移行』
- 『*Solaris* 共通デスクトップ環境 上級ユーザ及びシステム管理者ガイド』
- オンライン・ヘルプ・ボリューム

注 - 『*Solaris* 共通デスクトップ環境 上級ユーザ及びシステム管理者ガイド』には、アプリケーションをデスクトップに統合する際に役立つ情報が含まれています。

カレンダ API とスケジュール API の詳細は、X.400 API Association から XAPIA 規格の最新版を入手してください。住所は下記のとおりです。

X.400 API Association, 800 El Camino Real, Mountain View, California, 94043

開発環境用マニュアル

この節では、開発者向けマニュアル (『*Solaris* 共通デスクトップ環境 プログラマーズ・ガイド』を除く) の概要を示します。このマニュアルの他に、開発環境用のマニュアルには次のマニュアルが含まれています。

- 『共通デスクトップ環境 スタイル・ガイド』
- 『共通デスクトップ環境 アプリケーション・ビルダ・ユーザーズ・ガイド』
- 『共通デスクトップ環境 プログラマ概要』
- 『*Solaris* 共通デスクトップ環境 *Motif* への移行』
- 『共通デスクトップ環境 プログラマーズ・ガイド(ヘルプ・システム編)』
- 『共通デスクトップ環境 *ToolTalk* メッセージの概要』
- 『共通デスクトップ環境 プログラマーズ・ガイド(国際化対応編)』
- 『共通デスクトップ環境 *Dtksh* ユーザーズ・ガイド』
- 『*Common Desktop Environment: Product Glossary*』
- オンライン・マニュアル・ページ

『共通デスクトップ環境 プログラマ概要』

このマニュアルは、2 つのパートから構成されています。パート I には、実行時と開発環境の両方に関するハイレベルの情報など、共通デスクトップ環境のアーキテクチャの概要が含まれています。パート II には、アプリケーションを開発する前に知っておくべき情報と、開発環境のコンポーネントの説明があります。

このマニュアルは、共通デスクトップ環境の開発環境と開発者マニュアル・セットの概要を示すものです。アプリケーションの設計と開発を始める前に、まずこのマニュアルを読んでください。

『共通デスクトップ環境 スタイル・ガイド』

このマニュアルは、アプリケーション設計のスタイルのガイドラインと、共通デスクトップ環境のアプリケーション・レベルの認定の要件を説明します。これらの要

件は、Motif バージョン 2.1 の要件に共通デスクトップ環境固有の要件を追加したものです。

チェックリストでは、モデル・キーボードの形式を使用して、キーについて説明しています。チェックリストは、英語ロケールで左から右へ書かれる言語を対象としたアプリケーションを設計することを前提としています。キーボード入力を示す箇所では、Motif のモデル・キーボードの文字でキーが示されています。マウス・ボタンは、マウスのボタンの数に依存しない動作を示すために、仮想ボタン名称を使用して説明されています。

このマニュアルは、アプリケーション設計者が一貫性のあるアプリケーションを開発し、アプリケーション内の動作に一貫性を持たせるために役立つ情報を提供します。

『共通デスクトップ環境 アプリケーション・ビルダ・ユーザーズ・ガイド』

共通デスクトップ環境のアプリケーション・ビルダ (AppBuilder とも呼ばれます) は、共通デスクトップ環境アプリケーションを開発するための対話型ツールです。このツールは、アプリケーションのグラフィカル・ユーザ・インタフェース (GUI) の構築と、デスクトップの多くの便利なデスクトップ・サービス (ヘルプ、ToolTalk™、およびドラッグ & ドロップなど) の組み込みとを容易にする機能を提供します。このマニュアルでは、パレットから「オブジェクト」をドラッグ & ドロップしてインタフェースを作成する方法を説明します。また、インタフェース内のオブジェクト間の接続方法、アプリケーション・フレームワーク・エディタを使用してデスクトップ・サービスとの統合を簡単にする方法、C コードの生成方法、および AppBuilder 出力にアプリケーション・コードを追加して最終的なアプリケーションを生成する方法についても説明しています。

『共通デスクトップ環境 プログラマーズ・ガイド *(*ヘルプ・システム編*)*』

このマニュアルは、アプリケーション・ソフトウェアのためのオンライン・ヘルプの開発方法について説明しています。ヘルプ・トピックの作成方法と、オンライン・ヘルプを Motif アプリケーションに統合する方法が述べられています。

このマニュアルの対象読者は、次のとおりです。

- オンライン・ヘルプ情報の設計、作成、および表示する設計者

- 完全に統合されたヘルプ機能を提供するアプリケーション・ソフトウェアを作成する開発者

このマニュアルは、4 つのパートから構成されています。パート I では、アプリケーションのヘルプを設計するために設計者と開発者とが協力して行う役割について説明しています。パート II は、オンライン・ヘルプを構成および記述する設計者に必要な情報を説明しています。パート III は、ヘルプ・システムのアプリケーション・プログラマのツールキットを説明しています。パート IV は、国際化対応環境向けのオンライン・ヘルプの作成について、設計者とプログラマに必要な情報を説明しています。

『共通デスクトップ環境 *ToolTalk* メッセージの概要』

このマニュアルでは、メディア交換およびデスクトップ・サービスのメッセージ・セットの規則に準拠したアプリケーションを作成するための便利なルーチンとして提供される ToolTalk のコンポーネント、コマンド、およびエラー・メッセージについて説明しています。このマニュアルは、ToolTalk サービスを使用して他のアプリケーションと相互運用するアプリケーションを作成または保守する開発者のためのマニュアルです。

このマニュアルでは、一般的な ToolTalk の機能については説明していません。ToolTalk サービスの詳しい説明は、『*Solaris ToolTalk* リファレンスマニュアル』を参照してください。ToolTalk をより簡単に使用するには、『*Solaris ToolTalk and Open Protocols: Inter-Application Communication*』を参照してください。

『共通デスクトップ環境 プログラマーズ・ガイド (国際化対応編)』

このマニュアルは、アプリケーションを簡単にローカライズして、さまざまな言語と文化的規則を一貫したユーザ・インタフェースでサポートできるようにする、アプリケーションの国際化対応について説明しています。

特に、次の情報を提供しています。

- 開発者に対し、世界中で使用できるようなアプリケーションを書くためのガイドラインとヒントを提供しています。
- デスクトップのさまざまな階層にまたがる国際化トピックの全体像を提供しています。

- 参考資料および詳しい記述のあるマニュアルを示しています。標準の規格文書を参照する場合もあります。

このマニュアルは、既存の参考資料または概念的なドキュメントの説明をそのまま掲載するのではなく、特定の国際化トピックに関するガイドラインと規則を説明するものです。国際化トピックに焦点を置くものであり、オープン・ソフトウェア環境の中の特定のコンポーネントや階層について説明したものではありません。

『共通デスクトップ環境 ***Dtksh*** ユーザーズ・ガイド』

このマニュアルでは、デスクトップ Korn シェル (`dtksh`) スクリプトで Motif アプリケーションを作成する方法を説明しています。開発者が作業を始めるにあたって必要な基本的な情報に加え、徐々に複雑になるスクリプトの例を示しています。

このマニュアルは、特定の作業に適したシェル形式のスクリプト環境を探している開発者を対象としています。Korn シェル・プログラミング、Motif、Xt イントリンシクスの知識と、Xlib についてのある程度の知識があることを前提としています。

『***Solaris*** 共通デスクトップ環境 ***Motif*** への移行』

このマニュアルは、アプリケーション開発のために Solaris の Motif を使用する方法と、OPEN LOOK または Motif アプリケーションを Solaris CDE へ移植する方法について説明します。

『***Common Desktop Environment: Product Glossary***』

このマニュアルは、共通デスクトップ環境で使用する用語の包括的なリストです。この用語集は、デスクトップのすべてのユーザにとって、ソースおよび参照の基本となります。この用語集の読者は、エンドユーザ、開発者、翻訳者まで多岐にわたるため、読者や、用語の由来、グラフィカル・ユーザ・インタフェース (GUI) でその用語を使用する CDE コンポーネントについての情報も、用語定義の書式に含まれています。

Sun のマニュアルの注文方法

専門書を扱うインターネットの書店 Fatbrain.com から、米国 Sun Microsystems™, Inc. (以降、Sun™ とします) のマニュアルをご注文いただけます。

マニュアルのリストと注文方法については、`http://www1.fatbrain.com/documentation/sun` の Sun Documentation Center をご覧ください。

Sun のオンラインマニュアル

`http://docs.sun.com` では、Sun が提供しているオンラインマニュアルを参照することができます。マニュアルのタイトルや特定の主題などをキーワードとして、検索をおこなうこともできます。

表記上の規則

このマニュアルでは、次のような字体や記号を特別な意味を持つものとして使用します。

表 P–1 表記上の規則

字体または記号	意味	例
`AaBbCc123`	コマンド名、ファイル名、ディレクトリ名、画面上のコンピュータ出力、コード例を示します。	`.login` ファイルを編集します。 `ls -a` を使用してすべてのファイルを表示します。 `system%`
`AaBbCc123`	ユーザーが入力する文字を、画面上のコンピュータ出力と区別して示します。	`system%` **`su`** `password:`
AaBbCc123	変数を示します。実際に使用する特定の名前または値で置き換えます。	ファイルを削除するには、`rm` *filename* と入力します。

表 P–1 表記上の規則 続く

字体または記号	意味	例
『 』	参照する書名を示します。	『コードマネージャ・ユーザーズガイド』を参照してください。
「 」	参照する章、節、ボタンやメニュー名、強調する単語を示します。	第 5 章「衝突の回避」を参照してください。 この操作ができるのは、「スーパーユーザー」だけです。
\	枠で囲まれたコード例で、テキストがページ行幅を超える場合に、継続を示します。	`sun% grep '^#define \` `XV_VERSION_STRING'`

ただし AnswerBook2™ では、ユーザーが入力する文字と画面上のコンピュータ出力は区別して表示されません。

コード例は次のように表示されます。

- C シェルプロンプト

```
system% command y|n [filename]
```

- Bourne シェルおよび Korn シェルのプロンプト

```
system$ command y|n [filename]
```

- スーパーユーザーのプロンプト

```
system# command y|n [filename]
```

[ ] は省略可能な項目を示します。上記の例は、*filename* は省略してもよいことを示しています。

| は区切り文字 (セパレータ) です。この文字で分割されている引数のうち 1 つだけを指定します。

キーボードのキー名は英文で、頭文字を大文字で示します (例: Shift キーを押します)。ただし、キーボードによっては Enter キーが Return キーの動作をします。

ダッシュ (-) は 2 つのキーを同時に押すことを示します。たとえば、Ctrl-D は Control キーを押したまま D キーを押すことを意味します。

一般規則

- このマニュアルでは、英語環境での画面イメージを使っています。このため、実際に日本語環境で表示される画面イメージとこのマニュアルで使っている画面イメージが異なる場合があります。本文中で画面イメージを説明する場合には、日本語のメニュー、ボタン名などの項目名と英語の項目名が、適宜併記されています。
- このマニュアルでは、「IA」という用語は、Intel 32 ビットのプロセッサアーキテクチャを意味します。これには、Pentium、Pentium Pro、Pentium II、Pentium II Xeon、Celeron、Pentium III、Pentium III Xeon の各プロセッサ、および AMD、Cyrix が提供する互換マイクロプロセッサチップが含まれます。

パート I　基本的な統合方法

■ 第 1 章では、基本的な統合方法および印刷機能について説明します。

第 1 章

基本的なアプリケーションの統合方法

基本的なアプリケーションの統合方法は、実行することが強く推奨されるタスクのセットです。

- 23ページの「基本的な統合方法の特徴」
- 25ページの「基本的な統合方法の作業」

基本的な統合方法には、デスクトップのアプリケーション・プログラマ・インタフェース（API）の拡張使用は含まれません。したがって、ドラッグ&ドロップ、セッション管理、ToolTalk メッセージ、アクションおよびデータ型データベースへのプログラムのアクセスなど、デスクトップとの他の相互作用は提供されません。

この章で説明する統合方法の作業の一部は、ソースコードの変更を必要とします。それらはオプションですが、基本的な統合方法の作業と密接な関係があるため、ここで説明します。

基本的な統合方法の特徴

基本的なアプリケーションの統合方法には、エンドユーザ向けの次のような特徴があります。

- デスクトップ上のアプリケーションを探して起動するためのグラフィカルな方法

  アプリケーションはデスクトップ登録パッケージを提供し、インストール・スクリプトはアプリケーションを自動的に登録します。

  登録によって、アプリケーション・マネージャのトップ・レベルにアプリケーション・グループが作成されます。アプリケーション・グループにはアイコン

があり、ユーザがアイコンをダブルクリックすると、アプリケーションが起動します。

- アプリケーションのデータ・ファイルを認識し、操作する機能

  アプリケーションは、データ・ファイルにデータ型を提供します。

  データ型定義は、データ・ファイルが固有のアイコンを使うように構成して、データ・ファイルを見分けやすくします。また、データ・ファイルは、意味があるデスクトップ動作も持っています。たとえば、ユーザは、データ・ファイルをダブルクリックすることによってアプリケーションを起動したり、デスクトップのプリンタ・ドロップ領域にデータ・ファイルをドロップして、適切な印刷コマンドを使用してファイルを印刷できます。

- スタイル・マネージャによる簡単なフォントとカラーの選択

  アプリケーションは、インタフェースのフォントと、バックグラウンド、フォアグラウンド、およびシャドウの色を動的に変更します。

  デスクトップは、対応するアプリケーション固有のリソースがない場合に使われる一般的なインタフェースのフォントおよびカラーのリソースを定義します。

基本的な統合方法では、システム管理者に次のような利点を提供します。

- インストールと登録が容易

  インストール時に、アプリケーションは自動的に登録されます。他にシステム管理者がしなければならないことはほとんどありません。

- 運用時の管理が容易

  デスクトップの構成ファイルはすべて、一カ所に集められます。また、たとえば管理者がアプリケーションを更新したい場合や、別のアプリケーション・サーバに移動したい場合には、アプリケーションの登録解除が簡単にできます。

基本的な統合方法情報の構成

基本的な統合に必要な作業の大部分は、既存のアプリケーションをデスクトップに統合するシステム管理者によって実行されます。したがって、基本的な統合方法の説明の大部分は、『*Solaris* 共通デスクトップ環境 上級ユーザ及びシステム管理者ガイド』の第 5 章「アプリケーションの登録」にあります。

この章では、その説明の概要を紹介し、アプリケーション・プログラマ向けの追加説明をします。

基本的な統合方法の作業

基本的な統合に必要な一般的な作業は次のとおりです。

- フォントとカラーを設定するアプリケーションのリソースを変更する。これによってユーザは、スタイル・マネージャを使ってアプリケーションのインタフェース・フォントおよびカラーを変更できます。

  『*Solaris* 共通デスクトップ環境 上級ユーザ及びシステム管理者ガイド』の第5章「アプリケーションの登録」の「フォント・リソースおよびカラー・リソースの変更」の節を参照してください。

- アプリケーションのための登録パッケージを作成する。

  32ページの「アプリケーションのための登録パッケージの作成」と、『*Solaris* 共通デスクトップ環境 上級ユーザ及びシステム管理者ガイド』の第5章「アプリケーションの登録」を参照してください。

- 登録パッケージ・ファイルをインストールし、登録手順を実行するように、アプリケーションのインストール・スクリプトを変更する。

  『*Solaris* 共通デスクトップ環境 上級ユーザ及びシステム管理者ガイド』の第5章「アプリケーションの登録」の「`dtappintegrate` を使用したアプリケーションの登録」の節を参照してください。

- ネットワークおよびローカル・プリンタでアプリケーション・データ・ファイルを印刷する。デスクトップ・プリンタ・モデルは、印刷のためのグラフィカルな方法をユーザに提供し、UNIX の `lp` サービスの本来のネットワーク機能の上に構築されます。

印刷統合のレベル

ユーザが使用できる印刷機能は、統合のレベルによって異なります。統合には、次の3つのレベルがあります。

- 完全な統合。

  アプリケーションのソースコードを変更する能力がある場合には、完全な統合を行なってください。

  完全な印刷統合を行うと、ユーザはデータ・ファイルをプリンタ・ドロップ領域(フロントパネルのプリンタとプリント・マネージャのプリンタ・アイコン)にド

ロップすることによって、さまざまなプリンタで印刷できます。その他の特定のデスクトップ動作も実現します (26ページの「デスクトップ印刷環境変数」を参照してください)。

- 部分的な統合。

  アプリケーションのソースコードを変更する能力はないが、アクションによって印刷機能を呼び出すことができる場合には、部分的な統合を行なってください。

  部分的な統合をした場合、アプリケーションは完全統合機能の一部分の機能を提供します。たとえば、LPDEST 環境変数を使用することによって、アプリケーションの印刷機能は、印刷の出力先をドロップ領域から獲得します。

- 統合なし。

  アプリケーションがデータ・ファイルの印刷アクションを提供できない場合には、ユーザがファイルをプリンタ・ドロップ領域にドロップしたときにエラー・ダイアログ・ボックスを表示するように、データ・ファイルを構成しなければなりません。

完全な印刷統合

完全な印刷統合をするためには、アプリケーションは次の条件を備えていなければなりません。

- [印刷] アクションを提供する。
- 4 つのデスクトップ印刷環境変数を使用する。

デスクトップ印刷環境変数

完全に統合された印刷機能を持つためには、アプリケーションは、次の 4 つの環境変数を参照しなければなりません。LPDEST 変数は、とくに重要です。これによってユーザは、特定のプリンタ・ドロップ領域を使用して印刷の出力先を選択できます。

印刷環境変数	説明
LPDEST	指定された値をファイルのプリンタ出力先として使います。この変数が設定されていない場合には、アプリケーションのデフォルトの印刷装置が使用されます。
DTPRINTUSERFILENAME	印刷ダイアログまたは印刷出力に表示されるファイルの名前を指定します。この変数が設定されていない場合には、実際のファイル名が使用されます。
DTPRINTSILENT	印刷ダイアログ・ボックスを表示するかどうかを指定します。この変数に True が設定されているときには、印刷ダイアログは表示されません。この変数が設定されていない場合には、印刷ダイアログ・ボックスが表示されます。
DTPRINTFILEREMOVE	この変数に True が設定されているときには、印刷したあと、そのファイルは削除されます。この機能は、印刷後は保存しておく必要がない一時ファイルを対象としています。この変数が設定されていない場合には、ファイルは削除されません。

完全に統合された印刷アクション

印刷アクションは、登録パッケージの一部分であり、構成ファイル *app_root*/dt/appconfig/types/<language>/*name*.dt の中で提供されます。

印刷アクションが、26ページの「デスクトップ印刷環境変数」で示した 4 つの環境変数を参照するプログラムを実行する場合には、データ型は完全に統合されています。印刷アクションは、アプリケーションのデータ型に応じて書かなければならず、1 つのファイルだけを受け入れなければなりません。

たとえば、次の印刷アクションは、ThisAppData という名前のデータ型に固有です。

```
ACTION Print
{
 ARG_TYPE  ThisAppData
 EXEC_STRING  print_command -file %(file)Arg_1%
}
```

アプリケーションが ToolTalk の印刷要求を処理する場合には、印刷アクションは、次のアクションの変形で送ることができます。(4 つの環境変数のどれかが設定されていない場合には、対応するメッセージ引き数はヌルになります。メッセージ引き

数がヌルのときのデフォルトの解釈については、26ページの「デスクトップ印刷環境変数」を参照してください。)

```
ACTION Print
{
 ARG_TYPE   ThisAppData
 ARG_CLASS  FILE
 ARG_COUNT  1
 TYPE    TT_MSG
 TT_CLASS   TT_REQUEST
 TT_SCOPE   TT_SESSION
  TT_OPERATION  Print
 TT_FILE   %Arg_1%
 TT_ARG0_   MODE TT_IN
 TT_ARG0_   VTYPE %Arg_1%
 TT_ARG1_   MODE TT_IN
 TT_ARG1_   VTYPE LPDEST
 TT_ARG1_VALUE  $LPDEST
 TT_ARG2_MODE  TT_IN
 TT_ARG2_VTYPE  DTPRINTUSERFILENAME
 TT_ARG2_VALUE  $DTPRINTUSERFILENAME
 TT_ARG3_MODE  TT_IN
 TT_ARG3_VTYPE  DTPRINTSILENT
  TT_ARG3_VALUE  $DTPRINTSILENT
 TT_ARG4_MODE  TT_IN
  TT_ARG4_VTYPE  DTPRINTFILEREMOVE
 TT_ARG4_VALUE  $DTPRINTFILEREMOVE
}
```

```
ACTION Print
{
 ARG_TYPE       ThisAppData
 ARG_CLASS     BUFFER
 ARG_COUNT     1
 TYPE          TT_MSG
 TT_CLASS       TT_REQUEST
 TT_SCOPE       TT_SESSION
 TT_OPERATION   Print
 TT_ARG0_MODE   TT_IN
 TT_ARG0_VTYPE   %Arg_1%
 TT_ARG0_VALUE   %Arg_1%
 TT_ARG1_MODE   TT_IN
 TT_ARG1_VTYPE   LPDEST
 TT_ARG1_VALUE   $LPDEST
 TT_ARG2_MODE   TT_IN
 TT_ARG2_VTYPE   DTPRINTUSERFILENAME
 TT_ARG2_VALUE   $DTPRINTUSERFILENAME
 TT_ARG3_MODE   TT_IN
 TT_ARG3_VTYPE   DTPRINTSILENT
 TT_ARG3_VALUE   $DTPRINTSILENT
 TT_ARG4_MODE   TT_IN
 TT_ARG4_VTYPE   DTPRINTFILEREMOVE
 TT_ARG4_VALUE   false
}
```

フィルタされたデータまたは印刷の準備ができているデータのための印刷アクションの作成

デスクトップ印刷ユーティリティ /usr/dt/dtlp は、lp サブシステムに基づく機能を提供します。lp の印刷オプションを集めて、指定されたファイルを印刷します。

次の条件のどちらかに該当する場合には、アプリケーションは dtlp を使用できます。

- プリンタに送る前にデータ・ファイルを処理する必要がない。
- アプリケーションがデータ・ファイルを印刷できる形式に変換するためのフィルタを備えている。

dtlp の詳細は、dtlp(1)の マニュアル・ページを参照してください。

ファイルを印刷する準備ができている場合には、印刷アクションは、EXEC_STRING の中で dtlp を実行します。たとえば、次のようにします。

```
ACTION Print
{
  ARG_TYPE        ThisAppData
  EXEC_STRING        dtlp %Arg_1%
}
```

アプリケーションが変換フィルタを備えている場合には、dtlp を実行する前にフィルタが実行されなければなりません。たとえば、次のようにします。

```
ACTION Print
{
  ARG_TYPE    MyAppData
  EXEC_STRING     /bin/sh `cat %Arg_1%| filter-name | dtlp`
}
```

filter_name は、印刷フィルタ名です。

部分的な印刷統合

部分的な印刷統合をするためには、アプリケーションは、次のものを提供しなければなりません。

- 印刷アクション
- どの印刷環境変数がアクションによって処理されるかによって印刷が統合される範囲

部分的な統合のための印刷コマンドの提供

部分的な印刷統合を提供するためには、アプリケーションは、次の形式の印刷用コマンドを提供しなければなりません。

print_command [*options*] `-file` *filename*

options は、印刷環境変数のいくつかまたはすべてを参照する、あるいはどれも参照しないためのメカニズムを提供します (26ページの「デスクトップ印刷環境変数」を参照してください)。

この印刷用コマンドのもっとも単純な形式では、オプションを省略します。

print_command `-file` *filename*

このコマンド行を使うと、ユーザは、デスクトップのプリンタ・ドロップ領域を使用してアプリケーションのデータ・ファイルを印刷できます。ただし、印刷の出力先は、ドロップ領域によって設定されません。また、環境変数によって設定されたその他の印刷動作は実装されません。たとえば、デスクトップは直接サイレント印刷を行なったり、一時ファイルを削除したりできません。

印刷用コマンドでデスクトップ印刷環境変数に対応する別のコマンド行オプションを提供する場合には、別の統合を提供できます。

たとえば、次のコマンド行は `LPDEST` を参照する能力を提供します。

print_command
[`-d` *destination*] [`-file` *filename*]

destination は、出力先プリンタです。

次の印刷コマンド行は、4 つの変数すべてを参照するためのオプションを提供します。

print_command [`-d` *destination*] [`-u` *user_file_name*] [`-s`] [`-e`] `-file` *filename*

user_file_name	画面に表示されるファイル名
`-s`	サイレント印刷 ([印刷] ダイアログ・ボックスは表示されません。)
`-e`	印刷後にファイルは削除されます。

参照はアクション定義で発生します。詳細は、26ページの「デスクトップ印刷環境変数」を参照してください。

環境変数のコマンド行スイッチへの変換

アクションは 4 つの環境変数を参照できないが、対応するコマンド行オプションをとることができる場合について、この項では、環境変数をコマンド行オプションに変換する方法を説明します。

たとえば、次の例は LPDEST を参照する簡単な印刷アクションです。

```
ACTION Print
{
  ARG_TYPE     data_type
  EXEC_STRING     print_command -d $LPDEST -file %(file)Arg_1%
}
```

ただし、この印刷アクションは、LPDEST が設定されていない場合には予測できない動作をすることがあります。

変数が設定されていないときに適切な動作を提供する印刷アクションを作成するための 1 つの方法は、印刷アクションが使うシェル・スクリプトを作成することです。

たとえば、次のアクションとそれが使用するスクリプトは、4 つの環境変数すべてを正しく処理します。

```
ACTION Print
{
  ARG_TYPE     data_type
  EXEC_STRING    app_root/bin/envprint %(File)Arg_1%
}
```

envprint スクリプトの内容は次のとおりです。

```
#!/bin/sh
# envprint - sample print script
DEST=''''
USERFILENAME=''''
REMOVE=''''
SILENT=''''

if [ $LPDEST ] ; then
  DEST=''-d $LPDEST''
fi

if [ $DTPRINTUSERFILENAME ] ; then
  USERFILENAME=''-u $DTPRINTUSERFILENAME''
fi

DTPRINTFILEREMOVE=echo $DTPRINTFILEREMOVE | tr ``[:upper:]'' ``[:lower:]'''
if [ ``$DTPRINTFILEREMOVE'' = ``true'' ] ; then
  REMOVE=''-e''
```

```
fi

DTPRINTSILENT=`echo $DTPRINTSILENT | tr

``[:upper:]'' ``[:lower:]''` if [

``$DTPRINTSILENT'' = ``true'' ] ; then
  SILENT=''-s''
fi

print_command $DEST $USERFILENAME $REMOVE $SILENT -file $1
```

統合されていない印刷

アプリケーションがデスクトップと印刷機能を統合しない場合には、データ・ファイルを正しく印刷するために、ユーザがアプリケーションを開かなければなりません。

それでもやはり、アプリケーションのデータ・ファイルをプリンタ・ドロップ領域にドロップするときに動作する印刷アクションを提供するべきでしょう。さもなければ、デスクトップはファイルがテキスト・データを含むとみなして、印刷出力を勝手に行います。

デスクトップは、この目的のために [印刷なし] という印刷アクションを提供します。[印刷なし] アクションは、プリンタ・ドロップ領域を使用してデータ・ファイルを印刷できないことをユーザに知らせるダイアログ・ボックスを表示します。

[印刷なし] アクションは、 [印刷できません] ダイアログ・ボックスを表示します。

[印刷できません] ダイアログ・ボックスを使用するには、[印刷なし] アクションにマップされる、データ型に固有の印刷アクションを作成します。たとえば、アプリケーションのデータ型を次のように仮定します。

```
DATA_ATTRIBUTES MySpreadSheet_Data1
{
  ---
}
```

次の印刷アクションは、このデータ型の [印刷なし] にマップされます。

```
ACTION Print
{
  ARG_TYPE    MySpreadSheet_Data1
  TYPE     MAP
  MAP_ACTION   NoPrint
}
```

アプリケーションのための登録パッケージの作成

アプリケーションのために作成するデスクトップ登録パッケージは、アプリケーションのインストール・パッケージの一部にならなければなりません。登録

パッケージを作成するための手順は、既存のアプリケーションをデスクトップに統合するシステム管理者によって実行されます。これらの手順については、『*Solaris* 共通デスクトップ環境 上級ユーザ及びシステム管理者ガイド』の第 5 章「アプリケーションの登録」に説明があります。

パートII　推奨する統合方法

第 2 章から第 5 章では、次の推奨する統合作業を実行する方法を説明します。

- 標準フォントの別名を使用して、アプリケーションがデスクトップ準拠システム上で最も近いフォントを使用するようにする
- アプリケーションからエラー・メッセージを表示する
- セッション・マネージャと統合して、ログアウト時のアプリケーションの状態を保存する
- 既存の機能の直接操作アクセラレータとして、アプリケーションの中でドラッグ & ドロップを実現する
- 第 2 章「フォントの統合」
- 第 3 章「アプリケーションからのエラーの表示」
- 第 4 章「セッション・マネージャとの統合」
- 第 5 章「ドラッグ & ドロップとの統合」

第 2 章

フォントの統合

アプリケーションは、X 端末で、またはネットワークを介してリモート・ワークステーションで使用できます。このような状況では、ユーザの X ディスプレイを X Window System のサーバから使用できるフォントは、アプリケーションのデフォルトとは異なることもあり、使用できないフォントもあります。

共通デスクトップ環境 (CDE) によって定義された標準フォント名は、すべての CDE 準拠システムで使えるように保証されています。これらの名前は、実際のフォントを表すわけではありません。その代わりに、各システム・ベンダが、使用できるフォントの中で最も適したフォントにマップする別名です。アプリケーションの中でこれらのフォント名だけを使用する場合には、CDE 準拠システムで最もよく適合したフォントを使用できます。

標準アプリケーションフォント名が ISO Latin 言語環境でしか利用できないのに対して、これらの標準インタフェースフォント名はすべての言語環境で利用できることを保証されています。 詳細については、マニュアルページ、`DtStdInterfaceFontNames`、`DtStdAppFontNames` を参照してください。

標準インタフェースフォント

デフォルトのフォント名

標準的なインタフェースフォント名のセットは、表 2–2で説明する XLFD フィールド名の値によって定義されます。

表 2–1 標準インタフェースフォント名のフィールド名の値

フィールド	値	説明
FOUNDRY	dt	CDE 名
FAMILY_NAME	interface system あるいは interface user	CDE 標準インタフェースフォント名
WEIGHT_NAME	medium あるいは bold	フォントの太さ
SLANT	r	Roman
SET_WIDTH	normal	標準の太さ
SPACING	p あるいは m	
ADD_STYLE	size hint sans あるいは serif	xxs から xxl までのプロポーショナルまたはモノスペースの値 sans serif フォントに sans、または serif フォントに serif
PIXEL_SIZE	プラットフォーム依存	
POINT_SIZE	プラットフォーム依存	
RESOLUTION_X	プラットフォーム依存	
RESOLUTION_Y	プラットフォーム依存	

表 2–1 標準インタフェースフォント名のフィールド名の値 続く

フィールド	値	説明
AVERAGE_WIDTH	m	ユーザフォントのモノスペースフォント
	p	システムフォントのプロポーショナルフォント
NUMERIC FIELD	*	プラットフォーム依存
CHAR_SET_REGISTRY	iso8859-1	規格作成組織
ENCODING	1	文字セット番号

標準インタフェースフォントのポイント数

3 つのスタイルそれぞれの 7 つの名前付きポイント数が ADD_STYLE_NAME フィールドに付加されます。XLFD フォントのパターンは、数値の大きさではなく、名前付きサイズに一致します。これらの名前付きサイズが使われるのは、インタフェースフォントについては、その正確なサイズよりも名目上のフォントサイズが優先されるためです。また、個別に調整されたインタフェースフォントに起因する実装の差異によって、異なるシステム間に共通するポイント数を設定できないためです。

7 つのサイズ名は次の通りです。

xxs	extra extra small
xs	extra small
s	small
m	medium
l	large
xl	extra large
xxl	extra extra large

名前付けされたサイズを用いるのは、CDE が実行されるさまざまなモニタサイズと解像度に対応するだけの、十分なフォント数を提供するためです。また、ボタンラベル、ウィンドウ名などのさまざまなユーザー設定に GUI を適合させるためです。ただし、最も小さい xxl と最も大きい xxs は、一般的なディスプレイや X 端末を利用して CDE デスクトップで表示するのに最適なものとして用意されているもので、精度の低い印刷や見出しサイズのディスプレイには適しません。

標準インタフェースフォント名のパターン

XLFD パターンは、次のとおりです。

```
-dt-interface*-*
```

これらの値によって、XCDE 標準インタフェースフォント名のフルセットに論理的に適合します（特定の X サーバーの動作が暗黙に定義されるわけではないことに注意してください）。

例えば、西欧語のロケールでは、21の CDE 標準インタフェースフォント名が表示されます。

```
-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
```

7 つのシステムフォントサイズすべてについての、app-defaults ファイル内のパターンのフルセットは次のとおりです。

```
-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
```

これらのパターンはリソースファイルで使用することができ、すべての CDE 対応のシステム上の iso Latin-1 ロケール用の CDE 標準インタフェース名と一致します。詳細については、DtStdInterfaceFontNames(5) マニュアルページを参照してください。

CDE 構成ファイルでのフォントの使用

CDE は、すべてのプラットフォーム上の CDE で動作するアプリケーションで使用できる一般的な標準アプリケーション・フォント名のセットを、いくつかのサイズにおいて指定します。各 CDE ベンダは、標準フォント名のセットを使用可能なフォントにマップします。既存のフォントへのフォント名のマッピングは、ベンダによって異なります。

`app-defaults` ファイルの中で標準アプリケーション・フォント名を使用すると、すべての CDE プラットフォームで単一の `app-defaults` ファイルを使用できます。標準フォント名を使用しない場合には、各 CDE プラットフォーム上の各アプリケーションごとに別の `app-defaults` ファイルをそれぞれ提供しなければなりません。

すべての CDE システムは、13 の標準アプリケーション・フォント名のセットを、少なくとも 6 サイズで提供します。これは、12 の一般的なデザインと変形スタイル (serif および sans serif)、および記号フォントを表します。これらの標準名に加えて、特定の CDE プラットフォームに対応して標準名がマップされるフォント名が提供されます。また、追加の 4 つの標準フォント名 (固定幅フォント内に serif と sans serif の両方のデザインを可能にします) が CDE プラットフォーム・ベンダによって提供されることもあります。

これら 13 のフォント名は、ISO 8859-1 の文字セットを使用するロケールのために、CDE プラットフォームの中に用意されています。他のロケールでの標準フォント名の使用方法については、『*Solaris* 共通デスクトップ環境 プログラマーズ・ガイド (国際化対応編)』を参照してください。

標準アプリケーションフォント

デフォルトのフォント名

フォント名のセットは、表 2–2 に示されている XLFD フィールド名の値によって定義されます。

表 2–2　フォント名のフィールド名の値

フィールド	値	説明
FOUNDRY	dt	CDE 名
FAMILY_NAME	application	CDE 標準アプリケーション・フォント名
WEIGHT_NAME	mediumまたはbold	フォントの線の太さ
SLANT	r	ローマン
	i	イタリック
SET_WIDTH	normal	通常設定幅
ADD_STYLE	sans または	sans serif フォントまたは
	serif	serif フォント
PIXEL_SIZE	*	プラットフォーム依存
POINT_SIZE	*pointsize*	要求されたフォントのポイント・サイズ
RESOLUTION_X	*	プラットフォーム依存
RESOLUTION_Y	*	プラットフォーム依存
AVERAGE_WIDTH	p	プロポーショナル (システムフォント)
	m	固定幅 (ユーザフォント)
NUMERIC FIELD	*	プラットフォーム依存
CHAR_SET_REGISTRY	iso8859-1	規格作成組織
ENCODING	1	文字セット番号

標準名は、X Windows XLFD フォント命名スキーマに従って使用できます。プラットフォーム依存フィールドに対して適切なワイルドカードで正しく指定すれば、CDE フォント名は、有効な、対応するプラットフォーム依存フォントを確実に開きます。ただし、Xlib の `XListFont()` 関数の呼び出しから返される `XLFD` 名は、すべての CDE プラットフォーム上で同じであるとは限りません。

このような値を使うと、次の XLFD パターンは、特定のプラットフォーム上の CDE 標準 アプリケーション・フォント名のセットのすべてと一致します。

```
-dt-application-*
```

次のパターンは、CDE のボールドのプロポーショナルスペース・フォント (serif と sans serif の両方) に一致します。

```
-dt-application-bold-*-*-*-*-*-*-*-p-*-*-*-
```

また、次のパターンは、固定幅フォントに一致します (serif または sans serif、あるいは両方)。

```
-dt-application-*-*-*-*-*-*-*-*-m-*-*-*-
```

CDE 標準アプリケーション・フォント名のセットのすべては、次のように表すことができます。

```
-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
```

ポイント・サイズ

それぞれの標準アプリケーション・フォント名で使用できるポイント・サイズの完全なセットは、ベンダの CDE プラットフォームで出荷されるフォントのセットによって決まります (ビットマップ・フォントのみか、ビットマップ・フォントとスケーラブル・アウトライン・フォントの両方)。すべての CDE プラットフォーム上で使用できる必要最小限のサイズのセットは、X11R5 のデフォルトのマッピングを構成するビットマップ・フォントの標準サイズ (8、10、12、14、18、または 24) に対応します。

たとえば、単純な固定幅フォントの 6 サイズの全セットを次のパターンによって表すことができます。

```
-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
```

これらのパターンは、CDE プラットフォーム上の対応する標準フォント名に一致しますが、POINTSIZE 以外の数値フィールドはプラットフォームによって異なる場合があります。また、ベンダが標準名のセットを実装する方法によって、一致するフォントは serif か sans serif のどちらかになります。

app-defaults ファイル内の標準アプリケーション・フォント名

一つの app-defaults ファイルを作成してアプリケーションのフォント・リソースを指定し、それをすべての CDE プラットフォームで使用できます。定義される標準名の部分は、どのベンダのプラットフォームでも同じなので、app-defaults ファイルの中のリソース指定でこれらの値を指定できます。ただし、その他のフィールド (PIXEL_SIZE、RESOLUTION_X、RESOLUTION_Y、および AVERAGE_WIDTH) はプラットフォームによって異なることがあるので、ワイルドカードを使用しなければなりません。たとえば、appOne という名前のアプリケーションが必要とするデフォルトのリソースを指定するには、次のようにします。

```
appOne*headFont:
-dt-application-bold-r-normal-sans-*-140-*-*-p-*-iso8859-1

appOne*linkFont:
-dt-application-bold-i-normal-sans-*-100-*-*-p-*-iso8859-1
```

もう一つの例として、あるベンダのプラットフォーム上で動作する appTwo は、見出しとハイパーテキスト・リンクのために 2 つのフォント・リソースを定義すると仮定します。appTwo は、14 ポイントのボールドの serif フォント (Lucidabright bold) と 12 ポイントのボールドかつイタリックの sans serif フォント (Lucida bold-italic) を使用します。その場合、app-defaultsファイル内のフォント定義を、

```
apptwo *headingFont:
-b&h-lucidabright-bold-r-normal--20-140-100-100-p-127-iso8859-1

apptwo *linkFont:
-b&h-lucida-bold-i-normal-sans-17-120-100-100-p-96-iso8859-1
```

から

```
apptwo *headingFont:
-dt-application-bold-r-normal-serif-*-140-*-*-p-*-iso8859-1

apptwo *linkFont:
-dt-application-bold-i-normal-sans-*-120-*-*-p-*-iso8859-1
```

に変更します。他の CDE プラットフォーム上のフォント名がわからなくても、CDE 標準アプリケーション・フォント名で指定されたプラットフォームに独立したパターンは、各プラットフォーム上の適切なフォントを示します。

リソース定義の中で、*ワイルドカードを使用して示した例のように作成します。ポイント・サイズ以外の数値フィールドにワイルドカードを適用することによって、フォントの正確なピクセル・サイズまたは平均の幅が多少違っても、リソースがすべてのプラットフォーム上の CDE フォントに必ず一致するようにできます。

詳細は、`DtStdAppFontNames(5)` マニュアルページを参照してください。

第 3 章

アプリケーションからのエラーの表示

アプリケーションを実行しているユーザは、メッセージ・フッタ、エラー・ダイアログ、または警告ダイアログにメッセージが表示され、適宜、詳しい説明がオンラインヘルプにあることを期待します。共通デスクトップ環境のアプリケーションは、エラー・メッセージと警告を表示するための共通モデルに従います。

エラー・メッセージの表示方法

メッセージ・テキストの処理方法のために、ダイアログ、フッタ、または別のユーザ・インタフェースのどこかに表示しないと、ユーザはアプリケーションからのメッセージを見ることができません。

共通デスクトップ環境 (CDE) では、そのようなメッセージは、通常のユーザが定期的に調べることのないログ・ファイルに出力されます。警告、メッセージ、およびエラー条件を表示する場所を決めるときには、次の規則に従ってください。

- 情報を示すメッセージの場合は、アプリケーションのメッセージ・フッタにテキストを表示します。たとえば、「MyDoc ファイルがコピーされました」のようになります。

- エラーまたは重大な警告についてのメッセージの場合は(ユーザにとって重要な操作が失敗した場合のトラブルなど)、エラー・ダイアログまたは警告ダイアログに表示します。

エラー・ダイアログに表示する情報

優れたエラー・ダイアログまたは警告ダイアログでは、次の情報をユーザに提供します。

- 何が起きたか (ユーザの視点から)
- 原因 (ユーザが現在の作業と環境に関連づけて理解できるような簡単な表現で)
- 問題の解決方法

4、5 行のエラー・ダイアログで説明できない場合には、ダイアログにヘルプ・ボタンを追加して、ヘルプ・ボタンをアプリケーションのヘルプ・ボリューム内のトピックにリンクすることを検討してください。

メッセージの作成の詳細は、『*Solaris* 共通デスクトップ環境 プログラマーズ・ガイド(国際化対応編)』を参照してください。

メッセージ・ダイアログとオンライン・ヘルプのリンク

追加の背景情報が必要な場合や、4、5 行のダイアログではエラーを十分に説明できない場合には、オンライン・ヘルプにリンクするボタンを追加できます。

ダイアログのオンライン・ヘルプの追加は単純な作業です。特定のダイアログをオンライン・ヘルプの候補として決めたら、次の作業を実行します。

1. エラー・ヘルプに対して固有な **ID** を選択します。

   この ID が、オンライン・ヘルプ・テキストへのリンクとなります。ID は、64 文字以下でなければなりません。たとえば、`DiskSpaceError` のようになります。

2. ダイアログを作成して、ヘルプ・コールバックを追加します。

エラー・メッセージに対しては XmCreateErrorDialog()簡易関数を、警告に対しては XmCreateWarningDialog()簡易関数を使用して、次のようにヘルプ・コールバックを追加します。

```
XtAddCallback(dialog, XmNhelpCallback, helpfn, ``ID'');
```

この例では、*helpfn* はヘルプ・ダイアログを管理するために作成したヘルプ関数、文字列「ID」は、エラー・メッセージに対して選んだ ID です (たとえば、DiskSpaceError)。ヘルプ関数では、XmNlocationId リソースを ID の値に設定します。/usr/dt/examples/dthelp ディレクトリに、このようなヘルプ関数の設定例があります。

ヘルプ・ダイアログ・ウィジェットの作成と管理の詳細は、『*Solaris* 共通デスクトップ環境 プログラマーズ・ガイド*(*ヘルプ・システム編*)*』を参照してください。

3. エラー・メッセージに対応するヘルプ・セクションを書きます。

   ヘルプ・ボリュームの「メッセージ」の章に、メッセージの説明を書きます。ヘルプのソース・ファイルでは、メッセージごとにセクションを設けなければならず、セクションの始めの ID= 属性は、コードの中でエラーに対して選んだ ID と一致しなければなりません。

   たとえば、s1 セクション見出しでは、ID は DiskSpaceError です。

   次の見出しは、ユーザのシステムに十分なディスク領域がないときに、「ファイルを保存できません」というエラー・メッセージを表示します。

   ```
   <s1 ID=DiskSpaceError> ファイルを保存できません <\s1>
   ```

   規則によって、セクション見出しのテキストはエラー・ダイアログのテキストと 1 対 1 で対応しなければならないので注意してください。

4. ヘルプ・ファイルを再作成します。

   エラー・メッセージに対する新しいヘルプ・セクションは、ヘルプ・ファイルを再作成して (dthelptag プログラムを使用して)、アプリケーションを再コンパイルするとすぐにアクティブになります。

オンライン・ヘルプの記述と作成方法の詳細は、『*Solaris* 共通デスクトップ環境 プログラマーズ・ガイド*(*ヘルプ・システム編*)*』を参照してください。

回復処理ルーチン

エラー条件のための回復処理ルーチンがある場合には、ダイアログに [再実行] ボタンを追加することを検討してください。たとえば、システムのディスク空間が不足しているためにファイルをコピーできなかった場合、ダイアログに [再コピー] オプションがあれば、ユーザは、ディスク空間やアクセス権の問題を訂正してから、そのオプションを選択できます。

第 4 章

セッション・マネージャとの統合

セッション・マネージャは、ユーザが (現在のセッションから) ログアウトするときや、ユーザが (ホーム・セッションとして) 環境を保存するときに、デスクトップ環境と実行中のアプリケーションに関する情報を保存します。アプリケーションが現在のセッションまたはホーム・セッションの一部として保存され、次のセッションの一部として再起動されるためには、X クライアント間通信規約マニュアル (ICCCM) 1.1 のセッション管理プロトコルを理解できる必要があります。この章では、セッション・マネージャがセッションを保存して復元する方法を概説し、アプリケーションがセッション管理に関与するために必要な手順を詳しく述べます。

セッション・マネージャがセッションおよびアプリケーションを保存する方法

セッションを終了するときや、ホーム・セッションを保存するとき、セッション・マネージャは次の作業を実行します。

1. 選択されたリソース設定と X サーバ設定を保存する。
2. 各アプリケーションが状態を保存できるようにして、保存の完了を待つ。

3. アプリケーションの再起動に必要なコマンド行を獲得する。

セッション管理のためのアプリケーションのプログラム方法

プログラム環境の設定

この節では、統合プロセスの一部としてアプリケーションを保存するために必要なプログラミングの手順を説明します。

プログラム環境を設定するには、次の手順に従います。

1. 次のヘッダ・ファイルを組み込みます。
   - Xm/Xm.h
   - Xm/Protocols.h
   - Dt/Session.h
2. libXmと libDtSvcをリンクします。
3. ツールキットを初期化して、トップレベル・ウィジェットを作成します。

WM_SAVE_YOURSELF アトムの設定

次の例に示すように、Motif の XmAddWMProtocol() 関数を使用して、アプリケーションのトップレベル・ウィンドウの WM_PROTOCOLS 属性の WM_SAVE_YOURSELF アトムを設定します。

```
Atom XaWmSaveYourself;
Display *dsp;
dsp = XtDisplay(toplevel);
XaWmSaveYourself = XmInternAtom(dsp,
''WM_SAVE_YOURSELF'', False);

XmAddWMProtocols(toplevel, &XaWmSaveYourself, 1);
```

注 - 複数のウィンドウに対して WM_SAVE_YOURSELF アトムを設定しないでください。

WM_SAVE_YOURSELF メッセージを受け取るための準備

Motif の XmAddWMProtocolCallback() 関数を使用して、アプリケーションが WM_SAVE_YOURSELF クライアント・メッセージを受け取ったときに呼び出されるコールバック・プロシージャを設定します。

```
XmAddWMProtocolCallback(toplevel,
XaWmSaveYourself, SaveYourselfProc,
toplevel);
```

WM_SAVE_YOURSELF メッセージの処理

セッション・マネージャがこのアプリケーションのトップレベル・ウィンドウに WM_SAVE_YOURSELF クライアント・メッセージを送ると、SaveYourselfProc() コールバック・プロシージャが呼び出されます。このコールバックを使用して、アプリケーションの状態を保存します。アプリケーションはプログラマが選んだ任意の方法で状態を保存できますが、保存中はユーザと対話できません。

セッション・マネージャは、アプリケーションの状態を保存するための絶対パス名とベース・ファイル名を返す手段として、DtSessionSavePath() 関数を提供します。

WM_COMMAND 属性の設定

アプリケーションが WM_SAVE_YOURSELF メッセージの処理 (状態を保存するか、メッセージを無視する) を終了した後、アプリケーションはトップレベル・ウィンドウの WM_COMMAND 属性を設定して、保存操作が完了したことをセッション・マネージャに知らせなければなりません。

アプリケーションのトップレベル・ウィンドウの WM_COMMAND 属性を設定するには、Xlib の XSetCommand() 関数を使用します。この属性を設定することによって、アプリケーションが WM_SAVE_YOURSELF メッセージの処理を終了したことをセッション・マネージャに知らせ、アプリケーションを再起動するために必要なコマンド行をセッション・マネージャに与えます。

XSetCommand() は、コマンド引き数の配列を受け入れます。アプリケーションが保存プロセスの一部として DtSessionSavePath() 関数を使用する場合には、XSetCommand() には追加のコマンド引き数 -session *basename* が必要です。*basename* は、DtSessionSavePath() によって返されるベース・ファイル名です。

セッション・マネージャがセッションを復元する方法

セッション・マネージャは、次のようにしてセッションを復元します。

1. リソース・データベースとサーバ設定を復元する。
2. 保存されたコマンドを使用して、アプリケーションを再起動する。

アプリケーションが、保存された状態のパスを見つけるために `DtSessionSavePath()` を使用した場合には、アプリケーションは、ベース・ファイル名を`-session` 引き数から `DtSessionRestorePath()` 関数に渡して、保存状態ファイルの絶対パス名を見つけることができます。

第 5 章

ドラッグ & ドロップとの統合

この章では、ドラッグ & ドロップ・ユーザ・モデルと共通デスクトップ環境 (CDE) のドラッグ & ドロップ簡易アプリケーション・プログラム・インタフェース (API) を説明し、ドラッグ & ドロップの使い方を説明します。

概要

共通デスクトップ環境 (CDE) には、あらゆるデスクトップを通じて操作に一貫性のある便利なドラッグ & ドロップを提供するために、Motif に基づくドラッグ & ドロップのためのアプリケーション・プログラム・インタフェース (API) があります。CDE のドラッグ & ドロップ API は、開発者によるドラッグ & ドロップの実現をより簡単にします。ドラッグ & ドロップを使うと、ユーザは、画面上のオブジェクトをグラブし、ディスプレイ上をドラッグし、他のオブジェクトの上にドロップするという直接操作によって、データを転送できます。

テキスト、ファイル、およびバッファは、CDE のドラッグ & ドロップ API で使用されるデータの 3 つのカテゴリです。この文脈のテキストは、入力フィールドのテキストのように、ユーザの目に見えるテキストとして定義されます。ファイルは、ファイル・システム内にあるデータのコンテナです。各ファイルは、その内容を記述する形式を持ちます。バッファは、メモリに含まれるデータです。特徴として、各バッファはその内容を記述する形式を持ちます。

ライブラリとヘッダ・ファイル

ドラッグ & ドロップを使用するには、`DtSvc` ライブラリをリンクする必要があります。ヘッダ・ファイルは `Dt/Dnd.h` です。

デモ・プログラム

ドラッグ & ドロップの例が入っているデモ・プログラムは、`/usr/dt/examples/dtdnd` にあります。

ドラッグ & ドロップの使い方

ドラッグ & ドロップと統合するには

ドラッグ & ドロップとアプリケーションを統合するには、次の手順に従います。

1. **Dt/Dnd.h** を組み込みます。
2. **libDtSvc** をリンクします。
3. 受信側は次の作業を実行します。
   a. `DtDndDropRegister` を使用して、ドロップ領域を登録します。
   b. オプション。ドロップ・アニメーションのコールバックを書くこともできます。
   c. 転送コールバックを書きます。
4. 送信側は次の作業を実行します。

a. ユーザ・アクションを認識し (変換テーブルの変更が必要な場合があります)、DtDndDragStart を呼び出します。

b. 変換コールバックを書きます。

c. ドラッグ終了コールバックを書きます。

ドラッグ & ドロップ・ユーザ・モデル

この節では、デスクトップの他の部分に矛盾せずに、ユーザの期待に反しないアプリケーションが設計できるように、ドラッグ & ドロップの基本となるユーザ・モデルを説明します。

ドラッグ & ドロップの詳細と、ドラッグ & ドロップ要素の外観に関するガイドラインについては、『共通デスクトップ環境 スタイル・ガイド』を参照してください。

ドラッグ & ドロップをデスクトップ上のすべてのアプリケーションで使用できれば、システムはユーザにとってより予測可能なものとなり、したがって、より使いやすく覚えやすくなります。ユーザは、すでに知っている技術を使うことによって、自分が学んだことをより多くのアプリケーションに応用できます。また、多くのユーザはメニューを使うよりもドラッグ & ドロップを好みます。

この章では、ユーザが何かをドロップできる場所としてドロップ領域という用語を使用します。ドロップ領域は、通常、コントロールまたはグラフィック・アイコンによって表示されます。たとえば、ごみ箱アイコンや入力フィールドのグラフィックです。ドロップ領域を表す矩形の領域には、ドロップ・ターゲットという用語を使用します。

ドラッグ & ドロップ機能

ドラッグ & ドロップ機能があれば、ユーザはアイコンとして表されたオブジェクトを選択し、操作できます。

注 - ドラッグ & ドロップは、アプリケーション内でサポートされている他のユーザ・インタフェース・コントロールを通して使用できる機能のアクセラレータです。ただし、すべてのユーザがドラッグ & ドロップを利用できるわけではありません。基本的な操作は、ドラッグ & ドロップ以外にもサポート方法を用意してください。アプリケーションがドラッグ & ドロップを通してサポートする基本的な機能は、メニュー、ボタン、またはダイアログ・ボックスによってもサポートされなければなりません。

ドラッグ・アイコン

ユーザがドラッグ & ドロップを使用してアイコンを選択し、操作するときには、ドラッグされる項目を表すグラフィック・アイコンは、選択からドラッグ & ドロップの終了まで一貫していることをユーザは期待します。ユーザがファイル・マネージャのメッセージ・アイコンを選択してドラッグを開始した場合には、ドラッグ・アイコンの元の部分は、そのメッセージ・アイコンによって表されます。このような一貫性を与えることで、ドラッグ & ドロップはユーザにとって予測可能なものになります。転送先アプリケーションがアイコンを使用する場合、ほとんどのアイコンは、選択されてドラッグ & ドロップされたアイコンと同じでなければなりません。ただし、この動作は、すべてのアプリケーションで常に適切であるとは限りません。テキストのドラッグは例外です。選択されたテキストをドラッグする代わりに、テキスト・ドラッグ・アイコンが使用されます。

転送元と転送先の両方のアプリケーションが、ドラッグ・アイコンの外観を指定します。アプリケーションが一貫した適切なドラッグ・アイコンを持つようにするのは、開発者の責任です。ドラッグ & ドロップ・ライブラリはデフォルトのアイコンを提供しますが、各アプリケーションのために開発者が独自のアイコンを指定するとよいでしょう。アイコンとそのアイコンによって表されるデータ型を関連付けるために、データ型データベースを使用しなければならない場合があります。詳細は、第 9 章を参照してください。

ユーザがアイコンを選択せずにドラッグを開始する場合は、関連するドラッグ・アイコンを提供しなければなりません。たとえば、アポイント・エディタでは、ユーザはスクロール・リストからアポイントを選択できますが、アイコンが表示される場合と表示されない場合があります。ソース・インジケータとしてアポイント・アイコンを使用しなければなりません。転送先アプリケーション (たとえば、ファイル・マネージャ) は、同じアポイント・アイコンを表示しなければなりません。

ドラッグ・アイコンの各部

ドラッグ・アイコンがドロップ領域の上に来ると、ドラッグオーバ・フィードバックを提供するために外観が変化します。

ドラッグ・アイコンには次の 3 つの部分があり、その組み合わせによってドラッグオーバ・フィードバックを提供します。

- 状態インジケータ
- 操作インジケータ
- ソース・インジケータ

状態インジケータは、有効または無効ドロップ領域インジケータと組み合わされて、位置付けのために使用されるポインタです。有効状態インジケータは、矢印ポインタです。このポインタにはホット・スポットがあるので、ユーザは予測可能な方法で位置付けることができます。無効状態インジケータは、円と斜線の組み合わせであり、ユーザが無効なドロップ領域の上にカーソルを置いたときに表示されます。

操作インジケータは、ドラッグ時に行われる操作 (移動、コピー、またはリンク) に関するフィードバックをユーザに与えます。ほとんどのドラッグは移動なので、より頻度が少ないコピーまたはリンク操作を実行するときには、追加のフィードバックが与えられます。

注 - 操作フィードバックは、状態フィードバックとソース・フィードバックの手前に表示されます。この動作は、Motif のドラッグ & ドロップ動作と一致しています。

ユーザは、表 5–1 に示されている特定のキーを押しながらドラッグすることによって、ドラッグ操作 (移動、コピー、またはリンク) を選択できます。

表 5–1 ドラッグ操作を変更するためのキー

キー	操作
[Shift]	移動
[Control]	コピー
[Control]+[Shift]	リンク

ファイル・マネージャの読み取り専用ウィンドウの場合のように、転送元アプリケーションがコピーを強制することもあります。ユーザが操作を選択したとき

に、ドロップ領域がその操作と一致しなければドロップできません。一致しない場合には、ドロップ領域は無効です。つまり、ユーザが [Control] キーを押してコピーを選択して、ドラッグ・アイコンをごみ箱アイコンへドラッグした場合には、ごみ箱へのコピーは許可されません。このため、ドラッグ・アイコンはごみ箱アイコンを無効なドロップ領域として表示しなければならず、ドロップは失敗します。

ソース・インジケータは、選択 (すなわち、ドラッグされている項目) を表します。ソース・インジケータは、選択が 1 つの項目または複数の項目を表すか、あるいは選択が表す項目の種類によって変化します。

ウィンドウ内部からのドラッグ

アプリケーションは、ダイアログ・ボックスまたはウィンドウ内部からのドラッグを可能にする必要がある場合があります。カレンダのアポイントエディタには、アポイントのスクロール・リストとアポイントを編集するための入力領域があります。ユーザは、スクロール・リストからアポイントをドラッグできますが、アポイント入力領域からもドラッグできなければなりません。ユーザが入力領域からドラッグできるのは、アポイントがまだカレンダに挿入されていないときです (たとえば、申し込まれたミーティングの時間を入力したが、カレンダに挿入していないときなどです)。

ドラッグできる項目には、アイコン・グラフィックを関連付ける必要があります。ダイアログ・ボックスのグラフィック・アイコンは、ドラッグされる情報に隣接する適切な領域に置きます。ダイアログ・ボックスまたはウィンドウの右上隅が、望ましいデフォルトの位置です。このアイコンは何かをドラッグできることをユーザに知らせます。また、使用するグラフィックは、ドラッグ・アイコンに使用するグラフィックと同じにして、一貫性を持たせます。アイコンは 32 × 32 ピクセルでなければならず、ファイル・マネージャが使用するアイコンと同様のラベルがなければなりません。詳細は、『共通デスクトップ環境 スタイル・ガイド』の第 3 章「ドラッグ & ドロップ」を参照してください。

注 - ドラッグが可能なのは、選択できるコンポーネントまたは項目を持つヒューマン・インタフェース要素からだけです。ボタンまたはメニューのラベルなど、静的なラベルからはドラッグできません。

視覚的なフィードバック

この節では、ドロップ領域フィードバックとドラッグ & ドロップの遷移効果を説明します。

ドロップ領域フィードバック

デフォルトのドロップ領域フィードバックをドラッグアンダといい、領域を囲む実線、ドロップ領域を囲む斜角の付いた浮き出した表面かくぼんだ表面、またはドロップ領域の上に描かれたピックスマップで表されます。

遷移効果

遷移効果は、ドロップが成功したか失敗したかをユーザに知らせます。メルトとスナップバックという 2 つの遷移効果があります。

メルトは、ユーザがドラッグ・アイコンを有効なドロップ領域にドロップしたときに発生します。ユーザがドラッグ・アイコンを有効なドロップ領域にドロップすると、ドラッグ・アイコンは、ドロップ領域に溶けてなくなります。ドラッグ・アイコンは、転送先アプリケーションにふさわしいアイコンに置き換えられます。フロントパネルのプリンタは、メルト効果以外には何も示しません。開いているファイル・マネージャ・ウィンドウは、適切なアイコンを表示することがあります。

アイコンがドロップされても、メルト効果が直ちに起こらないこともあります。転送が完了するまで、アイコンが位置していた場所に表示されています。転送中は、転送先のカーソルをビジー状態に設定してください。転送が完了するまで、ユーザはアイコンを動かしたり、選択したりできません。ビジー・カーソルによって、転送中であることをユーザに知らせます。

スナップバックは、ドロップが失敗したときに発生します。ドロップの失敗には、2 通りあります。ユーザが無効なドロップ領域にドラッグ・アイコンをドロップした場合には、ドラッグ・アイコンは転送元アプリケーションへ戻ります (スナップバックします)。ドロップが発生したら、転送元と転送先のアプリケーションはデータを転送しなければなりません。データ転送が失敗した場合には、ドラッグ・アイコンはスナップバックし、転送先アプリケーションは失敗したことをユーザに通知し、ドロップが失敗した理由を示さなければなりません。

ドラッグ & ドロップの転送元 (ソース)

ドラッグ & ドロップの転送元の動作が理解できるように、表 5–2 に、選択されたテキスト、ファイル、およびバッファのドラッグ・ソースにできる主なデスクトップ・コンポーネントを示します。

表 5–2 ドラッグ・ソースにできるデスクトップ・コンポーネント

ドラッグ・ソース	選択されたテキスト	ファイル	バッファ
テキスト・フィールド (Motif)*	選択されたテキスト	N/A	N/A
テキスト・エディタ: メイン・ウィンドウ	選択されたテキスト	N/A	N/A
端末エミュレータ: メイン・ウィンドウ	選択されたテキスト	N/A	N/A
ファイル・マネージャ: フォルダ・ウィンドウ	N/A	ファイル	N/A
ファイル・マネージャ: ごみ箱ウィンドウ	N/A	ファイル	N/A
メール: メッセージ・リスト	N/A	N/A	メールメッセージ形式のメッセージ
メール: アタッチメント・リスト	N/A	N/A	アタッチメント形式のアタッチメント
カレンダ: アポイントエディタ	N/A	N/A	アポイント形式のアポイント

* Motif テキスト・フィールドの転送元が選択されたアプリケーションは、テキストをドラッグします。

ドラッグ & ドロップの転送先

次のデスクトップ・コンポーネントは、ドロップ先になります。

- エディタ
- ファイル・マネージャ
- フロントパネル

各コンポーネントは、選択されたテキスト、ファイル、およびバッファのドロップを受け入れます。テキスト・ドロップの転送先のほとんどは、Motif ライブラリによって自動的に提供されます。ファイルまたはバッファ・データのドロップ先がドロップを受け入れるためには、プログラムを追加しなければなりません。

ユーザがファイルからデータをドロップして、そのファイルが何らかの方法で変更されたときには、ファイルの元の保持者へ変更を書き戻すことができます。この動作をセーブバックといいます。ただし、データがバッファからドロップされたときには、データは元のファイルに関する情報を持ちません。つまり、データの元の保持者がないので、バッファからのデータに加えられた変更を書き戻すことはできません。この動作をセーブバックなしといいます。

たとえば、メール・プログラムは、ドラッグ & ドロップを使用して、メール・アタッチメントをエディタにエクスポートできます。アタッチメントがバッファとしてエクスポートされた (セーブバックがない) 場合、エディタでメール・プログラム内の元のアタッチメントを変更する手段はありません。したがって、エディタは、アタッチメントの変更済みの版を新しいファイルに保存するしかありません。

メール・アタッチメントは、すでに別のファイルではない (メール・フォルダ・ファイルに埋め込まれている) ので、バッファとしてエクスポートされるだけで、他のエディタによって保存できません。

アタッチメントがファイルとしてエクスポートされた (セーブバックがある) 場合には、エディタは変更済みのものを同じファイルに保存します。

表 5–3 は、テキスト・エディタ、アイコン・エディタ、カレンダ、メール・プログラムなどのエディタ型コンポーネントへ、選択されたテキスト、ファイル、およびバッファをドロップする場合を示します。

表 5–3 エディタのドロップ先

ドロップ先	選択されたテキスト	ファイル	バッファ
テキスト・エディタ: メイン・ウィンドウ	挿入	挿入	挿入
端末エミュレータ: メイン・ウィンドウ	挿入	N/A	N/A
アイコン・エディタ: メイン・ウィンドウ	N/A	読み込み (ファイルがアイコン形式の場合)。セーブバックあり	読み専用で読み込み (データがアイコン形式の場合)。セーブバックなし
メール・プログラム: メッセージ・リスト	N/A	追加 (ファイルがメール形式の場合)	追加 (データがメール形式の場合)
メール・プログラム: メール作成	挿入	挿入	挿入
メール・プログラム: アタッチメント・リスト	挿入	挿入	挿入
カレンダ: メイン・ウィンドウ	N/A	アポイントをスケジュール (ファイルがアポイント形式の場合)	アポイントをスケジュール (データがアポイント形式の場合)
カレンダ: アポイントエディタ	テキスト・フィールドへ挿入	アポイント・フィールドに記入 (ファイルがアポイント形式の場合)	アポイント・フィールドに記入 (データがアポイント形式の場合)
アプリケーション・ビルダ	N/A	読み込み (ファイルが BIX または BIL 形式の場合)。セーブバックあり	読み専用で読み込み (データが BIP 形式の場合)。セーブバックなし

表 5–4 は、ファイル・マネージャ内のファイルとフォルダ・アイコンへ、選択されたテキスト、ファイル、およびバッファをドロップする場合を示します。

表 5–4 ファイル・マネージャのドロップ先

ドロップ先	選択されたテキスト	ファイル	バッファ
ファイル・アイコン	ターゲット・ファイルとドロップされたテキストに対してドロップ・アクションを呼び出す (ファイルがテキストのドロップを受け入れ、ドロップされたテキストが適切な形式の場合)。セーブバックなし／コピーなし	ターゲット・ファイルとドロップされたファイルに対してドロップ・アクションを呼び出す (ファイルがファイルのドロップを受け入れ、ドロップされたファイルが適切な形式の場合)。セーブバックあり	ターゲット・ファイルとドロップされたデータに対してドロップ・アクションを呼び出す (ファイルがデータのドロップを受け入れ、ドロップされたデータが適切な形式の場合)。セーブバックなし／コピーなし
フォルダ・アイコン	テキストをフォルダ内の新しいファイルに「タイトルなし」という名前で挿入する	ファイルをフォルダにコピー／移動する	データをフォルダ内の新しいファイルに指定された名前 (指定された場合) で挿入する。名前が指定されなかった場合には、「タイトルなし」という名前で挿入する
アクション・アイコン	テキストに対してアクションを呼び出す (適切な形式であり、テキストのドロップを受け入れる場合)。セーブバックなし	ファイルに対してアクションを呼び出す (適切な形式であり、ファイルのドロップを受け入れる場合)。セーブバックあり	データに対してアクションを呼び出す (適切な形式であり、データのドロップを受け入れる場合)。セーブバックなし
メール・コンテナ・アイコン	メールボックスに追加する (テキストがメール形式の場合)	メールボックスに追加する (ファイルがメール形式の場合)	メールボックスに追加する (データがメール形式の場合)

表 5–5 は、フロントパネルのアクション・アイコンへ、選択されたテキスト、ファイル、およびバッファをドロップする場合を示します。

表 5–5　フロントパネルのドロップ先

ドロップ先	選択されたテキスト	ファイル	バッファ
テキスト・エディタ	読み専用で読み込む。セーブバックなし	読み込む。セーブバックあり	読み込み専用で読み込む。セーブバックなし
カレンダ	アポイントをスケジュールする (テキストがアポイント形式の場合)	アポイントをスケジュールする (ファイルがアポイント形式の場合)	アポイントをスケジュールする (データがアポイント形式の場合)
メール	テキストを接続してメッセージを作成する	ファイルを接続してメッセージを作成する	データを接続してメッセージを作成する
プリンタ	テキストを印刷する (印刷方法がテキストに対して有効な場合)	ファイルの内容を印刷する (印刷方法がファイル形式に対して有効な場合)	データを印刷する (印刷方法がデータの形式に対して有効な場合)
ごみ箱	N/A	ファイルをごみ箱へ移動する	N/A
サブパネル: アイコンのインストール	N/A	アイコンをインストールする	N/A
サブパネル: アクション	ファイル・マネージャと同じ	ファイル・マネージャと同じ	ファイル・マネージャと同じ
サブパネル:実行形式	ファイル・マネージャと同じ	ファイル・マネージャと同じ	ファイル・マネージャと同じ

ユーザに対するドラッグ & ドロップの表示方法とガイドラインの詳細は、『共通デスクトップ環境 スタイル・ガイド』を参照してください。

ドラッグ & ドロップ簡易 API

CDE は、デスクトップ内の一貫性と相互運用を促進し、開発者によるドラッグ & ドロップの実現を容易にするために、ドラッグ & ドロップ簡易 API を提供します。

ドラッグ & ドロップのための既存の Motif の API は、トランザクション中の転送元と転送先アプリケーションの通信を達成するための合理的な機能を提供します。データ転送のためのフレームワークを提供しますが、実際のデータ転送の詳細はアプリケーションに依存します。デスクトップ内のアプリケーション間の真の一貫性と相互運用のためには、すべてのアプリケーションが同じデータ転送プロトコルを使用しなければなりません。CDE のドラッグ & ドロップ簡易 API は、共通のデータ転送ルーチンを提供します。

開発者が簡単に使用できる

ドラッグ & ドロップのための既存の Motif の API は非常に柔軟性がありますが、その分、未熟な開発者にとっては使いにくい点もあります。CDE のドラッグ & ドロップ簡易 API は、次に説明するいくつかの簡易関数とサービスを提供することによって、より簡単で使いやすい API になっています。

- ドラッグ・アイコンの構成と外観を管理します。Motif のドラッグ・アイコンを構成するデフォルトのソース、状態、および操作アイコンのグラフィックが用意されています。これらのアイコンの組み合わせによって、ドラッグされているデータの型をチェックします。
- ドロップのアニメーションを可能にします。ドロップが完了したときに呼び出されるアニメーション・プロシージャを定義できます。
- テキスト、ファイル、およびバッファ転送のために、標準の X Window System の選択ターゲットを使用してデータ転送を提供します。このデータ転送は、標準のターゲットを直接使用する他のアプリケーションとの相互運用を可能にします。
- 二重登録を提供します。テキスト・ウィジェットをテキスト以外のデータのためのドロップ領域として登録できます。その場合でも、テキストのドロップを受け入れる機能は変わりません。

ポリシーの確立

ドラッグ & ドロップ API は、次の 3 つの分野のポリシーを確立します。

- 共通ターゲット。使用可能な場合には、クライアント間通信規約マニュアル (ICCCM) によって定義された既存の選択ターゲットが使用されます。
- データ転送プロトコル。API は、データ転送の詳細を隠し、データを単純なデータ構造体の形でアプリケーションに提示します。

- デフォルトのドラッグ・アイコン。デフォルトのドラッグ・アイコンは、それらを受け入れることができるアプリケーションのために用意されています。

共通の機能性の提供

ドラッグ & ドロップ API は、次の分野での共通の機能性を提供します。

- テキスト、ファイル名、およびバッファとしてのデータ転送をサポートします。
- データ転送フレームワークによって、新しい組み込みプロトコルの追加をサポートします。

既存の Motif API の応用

ドラッグ & ドロップのための API は、新しいドラッグ & ドロップ・サブシステムを使用するわけではなく、既存の Motif API を使用しています。また、共通のデータ転送プロトコルが選択されているので、使用可能な場合には、アプリケーションは新しい API をグローバルに使用しなくても、選択プロトコル・レベルで相互運用できます。

テキストとファイルの転送は、既存のプロトコルを使用します。バッファ転送は、新しいプロトコルを使用します。

ドラッグ & ドロップ処理

基本的なドラッグ & ドロップ処理の実行例を、図 5–1 に示します。

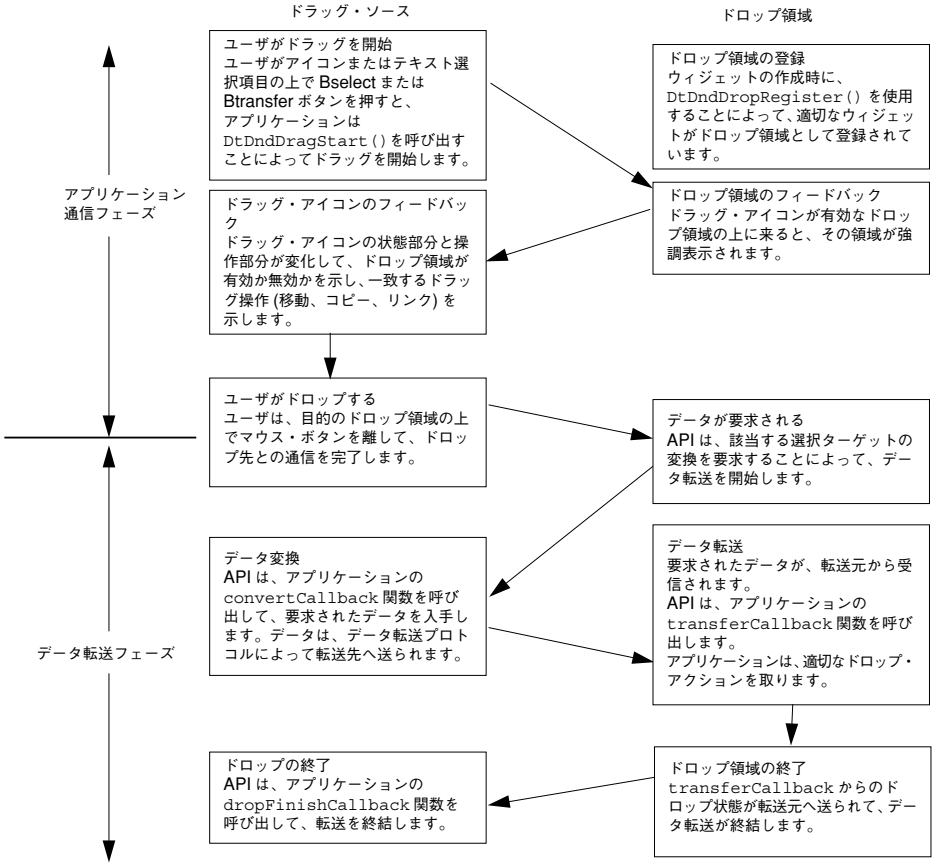

図 5–1　基本的なドラッグ & ドロップ処理

図 5–2 は、オプションのドラッグ & ドロップ処理の変化と操作を示します。破線のボックスは、基本的な処理を示します。実線のボックスは、オプションの変化と操作を示します。

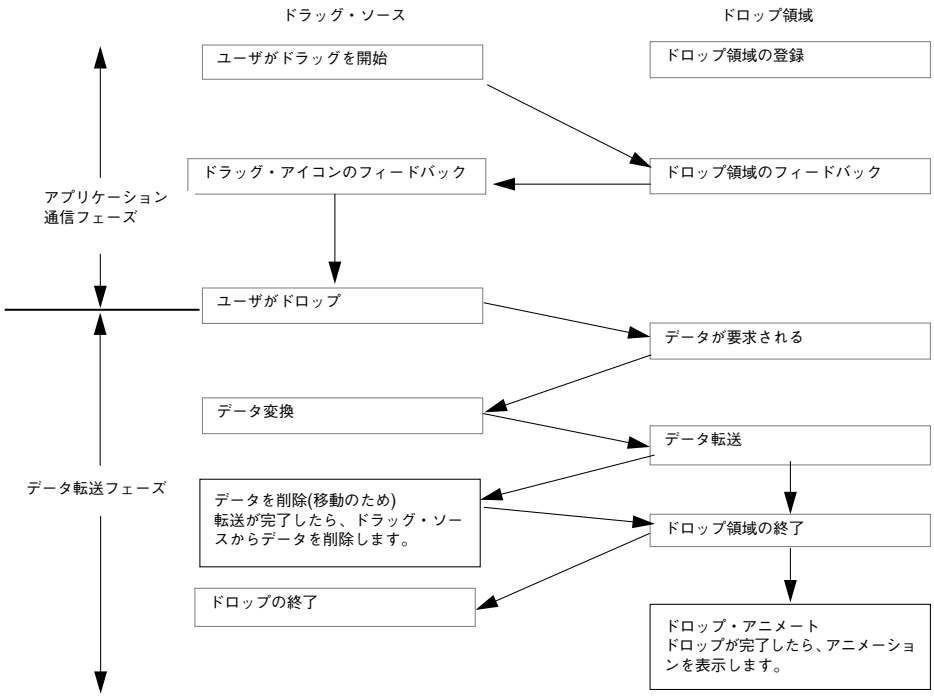

図 5–2　オプションのドラッグ & ドロップの変化と操作

統合アクション・プラン

この節では、アプリケーションと CDE のドラッグ & ドロップとの統合のためのアクション・プランを提案します。

ドラッグ & ドロップ API とサンプル・コードの検討

この章の説明を読んで、ドラッグ & ドロップ API を理解してください。API の基本的な理解ができたら、ドラッグ & ドロップのデモ・プログラム `/usr/dt/examples/dtdnd` のソースコードを見てください。このコードは、さまざまな API の使い方の例を提供しています。これらの例によって、アプリケーションでドラッグ & ドロップをサポートするために書かなければならないコードの性質と量が理解できます。アクションとデータ型 API の理解にも役立ちます。

可能なドロップ領域についてのアプリケーションの検討

アプリケーションがドラッグ & ドロップの処理を通して受け入れるデータの型を決めます。たとえば、ビットマップ・エディタを作成する場合には、ファイルのドロップをサポートしたいことがあります。アプリケーションにドロップできるデータ型を決めたら、ドロップ領域になるウィジェットを決めます。ビットマップ・エディタの例の場合には、ビットマップ編集領域を、アプリケーション上でファイルをドロップできる唯一の場所として決定できます。この場合、`DtDndDropRegister()` を使用して、この領域を表すウィジェットを登録し、適切なコールバックを提供します。

ファイル名のドロップの処理は最も簡単なので、ファイル名のドロップの実現から始めてください。この手法をマスターすると、簡単にテキストとバッファのドロップの実現へ進むことができます。

可能なドラッグ・ソースに関するアプリケーションの検討

アプリケーションがドラッグ & ドロップの処理の転送元として許可するデータの型を決めます。ビットマップ・エディタの例の場合、カット & ペーストのアクセラレータとして、現在のビットマップ選択を含んでいるビットマップ・データをドラッグ・ソースにしたいことがあります。アプリケーションからドラッグできるデータ型を決めたら、ドラッグ・ソースになるウィジェットを決めます。ビットマップ・エディタの例の場合、強調表示されているビットマップ選択を含んでいるビットマップ編集領域がドラッグ・ソースの役目を果たすと決定できます。この場合、この領域を表しているウィジェットがドラッグ・ソースになります。

アプリケーションに最もふさわしい、または特有のバッファのドラッグの実現から始めてください。また、アプリケーションの複数起動間の簡単なデータ転送を可能にするために、アプリケーションにバッファをドロップする機能を追加する必要がある場合もあります。

API の概要

この節では、ドラッグ & ドロップのアプリケーション・プログラム・インタフェース (API) の概要を説明します。

DtSvc ライブラリとヘッダ・ファイル

ドラッグ & ドロップ機能は、デスクトップ・サービス・ライブラリ DtSvc で実現されます。ドラッグ & ドロップ API にアクセスするには、ヘッダ・ファイル <Dt/Dnd.h> を組み込み、-lDtSvc をつけてリンクします。

関数

API には 4 つの関数呼び出しがあり、ヘッダ・ファイル Dnd.h の中で宣言されています。これらの関数について、以下の段落で概説します。これらの関数については、後の節で詳しく説明します。

- DtDndDragStart() は、ユーザ・アクションに応答して、ドラッグを開始します。
- DtDndCreateSourceIcon() は、DtDndDragStart() で使用するソース・アイコンを作成します。
- DtDndDropRegister() は、ウィジェットをドロップ領域として登録します。ドロップ領域の登録は、通常はウィジェットが作成された直後に行われますが、いつ行なってもかまいません。
- DtDndDropUnregister() は、以前に登録したウィジェットの登録を解除します。ドロップ領域は、通常は破壊される直前に登録解除されますが、登録後であれば、いつ登録解除してもかまいません。

DtDndContext 構造体

データ転送を処理するには、DtDndContext のデータ構造を使用します。この構造体には、転送プロトコルのためのフィールド、転送される項目数、および転送されるデータ項目の配列が入ります。この構造体の構文の詳細は、DtDndDragStart(3X) および DtDndDropRegister(3X) のマニュアル・ページを参照してください。

プロトコル

プロトコルは、転送データの型を API に知らせるために使用されます。定義済みプロトコルを表 5–6 に示します。

表 5–6　定義済みプロトコル

プロトコル	説明
DtDND_TEXT_TRANSFER	テキスト転送。コンパウンド・テキスト (Motif は、テキスト転送のためにコンパウンド・テキスト・ターゲットを使用します。)
DtDND_FILENAME_TRANSFER	ファイル名転送
DtDND_BUFFER_TRANSFER	メモリ・バッファ

操作

表 5–7 に示すように、ドラッグ・ソースとドロップ領域は 3 つの方法のいずれか 1 つでデータを転送できます。

表 5–7　データ転送操作

操作	説明
XmDROP_MOVE	データを移動します (コピーしてから削除します)。
XmDROP_COPY	データをコピーします。
XmDROP_LINK	データへのリンクを含みます。

ドラッグ・ソースの使い方

この節では、ドラッグ・ソースの使い方を説明します。

ドラッグの開始

ドラッグは、2 つの方法のどちらかで開始されます。1 つは、ユーザは、`Btransfer` (中央マウス・ボタン) を押すことで、ドラッグを開始できます。ボタンが押されるとすぐに、ドラッグが開始されます。もう 1 つは、ユーザは

Bselect (左マウス・ボタン)を押しながらカーソルを動かすことによって、ドラッグを開始できます。ユーザがマウスを特定の距離だけ動かすと、ドラッグが開始されます。この距離をドラッグしきい値といい、ピクセル単位で示されます。Bselect のデフォルトのドラッグしきい値は、10 ピクセルです。Btransfer のドラッグしきい値は 0 です。ドラッグしきい値がないので、ポインタを動かすとすぐに、ドラッグが開始されます。Motif スクロール・テキスト・リストおよびテキスト・ウィジェットは、Btransfer と Bselect によってテキスト・ドラッグするためのドラッグ・ソースとして自動的に登録されます。

リストまたはアイコンからのドラッグ

ドラッグ・ソースとして使用できる 2 つの一般的なインタフェース・オブジェクトがあります。すなわち、リストとアイコンです。Motif リスト・ウィジェットは、自動的にテキスト・ドラッグの転送元を示します。他の種類のドラッグが必要な場合には、デフォルトのウィジェット変換を新しい Btn1 と Btn2 変換で無効にすることによって行われます。Motif にはアイコン・ウィジェットはありませんが、通常は描画領域をアイコンのコンテナとして使用します。この場合、ドラッグを開始するために、Btn1Motion のイベント・ハンドラが使用されます。コーディング例の詳細は、/usr/dt/examples/dtdnd のサンプル・コードを参照してください。

ドラッグしきい値

Bselect を使用してドラッグを開始するときには、ウィジェット・イベント・ハンドラまたは変換手順は、ドラッグを開始する前に、10 ピクセルのドラッグしきい値を適用しなければなりません。Btransfer にしきい値はないので、直ちにドラッグが開始されます。

Btransfer または Badjust

スタイル・マネージャの [マウス] カテゴリには、Btn2 (中央マウス・ボタン) が Btransfer として機能するか、それとも Badjust として機能するかを制御する設定があります。この設定は、リソース名 enableBtn1Transfer として格納されます。1 の設定は、Btn2 が Badjust であり、選択を調節することを示します。他の値の設定は、Btn2 が Btransfer であり、ドラッグを開始することを意味します。Btn1 (左マウス・ボタン) は、常にドラッグを開始します。

次の例は、Btn2 が Btransfer か Badjust であるかを決める方法を示します。

```
Display* display;

int
adjust;
  XtVaGetValues ((Widget)XmGetXmDisplay(display,
 ``enableBtn1Transfer'', &adjust,  NULL);

  if (adjust == 1)
    /* Btn2 is adjust */
 else
      /* Btn2 is transfer */
```

ドラッグの開始

共通デスクトップ環境 1.0 アプリケーションは、DtDndDragStart() を呼び出すことによってドラッグを開始します。この関数は、ドラッグを開始するためにデスクトップに特有の準備を行い、XmDragStart() を呼び出します。DtDndDragStart() 関数の形式とパラメータの使用法は、次のとおりです。

```
Widget DtDndDragStart(
 Widget   dragSource,
 XEvent    *event,
  DtDndProtocol   protocol,
 Cardinal   numItems,
 unsigned char   operations,
 XtCallbackList  convertCallback,
 XtCallbackList dragFinishCallback ArgList   argList,
 Cardinal   argCount)
```

`Widget dragSource`

ドラッグを始めたイベントを受け取るウィジェット

`XEvent *event`

ドラッグを始めたボタンが押された、またはボタン・モーション・イベント

`DtDndProtocol protocol`

データ転送のために使用するプロトコル。プロトコルは、次のいずれか 1 つを使用できます。

`DtDND_TEXT_TRANSFER`

`DtDND_FILENAME_TRANSFER`

`DtDND_BUFFER_TRANSFER`

`Cardinal numItems`

ドラッグする項目数を指定します。

unsigned char operations

dragSource によってサポートされるオプションを指定します。オプションは、XmDROP_MOVE、XmDROP_COPY、および XmDROP_LINK です。ドラッグ・ソースは、これらの操作の任意の組み合わせをサポートできます。操作の組み合わせを指定するには、or (|) を使用します。たとえば、移動とコピー操作をサポートするには、XmDROP_MOVE | XmDROP_COPY と指定します。

XtCallbackList convertCallback

このコールバックは、ドロップが開始され、ドロップ領域がドラッグ・ソースからデータを要求したときに呼び出されます。convertCallback については、次の節で詳しく説明します。

XtCallbackList dragFinishCallback

このコールバックは、ドラッグ & ドロップ・トランザクションが完了したときに呼び出されます。dragFinishCallback は、dragMotionHandler() をリセットして、ドラッグ & ドロップ・トランザクション時にドラッグ・ソースによって割り当てられたメモリを解放します。

変換コールバックの使い方

変換コールバックは、ドロップが発生すると、ドロップ領域にデータを提供します。変換コールバックの最初のアクションは、callData の中の reason フィールドの確認です。reason が DtCR_CONVERT_DATA または DtCR_CONVERT_DELETE でない場合には、直ちに戻さなければなりません。そうでない場合には、データの変換を続けます。たとえば、ファイル名の変換を処理する場合には、内部のデータ構造体から該当するファイル名を検索して、ファイル・データ・オブジェクトにコピーします。ドラッグ・ソースが移動操作をサポートしている場合には、DELETE ターゲットの変換をサポートする必要があります。すなわち、reason が DtCR_CONVERT_DELETE で convertCallback が呼ばれた場合は、移動されたデータに対して適切な削除アクションを実行します。ファイル転送の場合には、ファイルを削除します。次に、ファイル名の変換と削除を処理する簡単な convertCallback を示します。

```
void
convertFileCallback(
```

```
  Widget dragContext,
  XtPointer clientData,
  XtPointer callData)
{
  DtDndConvertCallbackStruct *convertInfo = (DtDndConvertCallbackStruct*)
  allData;
  char  *fileName = (char *) clientData;
   if (convertInfo->reason == DtCR_DND_CONVERT_DATA)
   {
  convertInfo->dragData->data.files[0]=
     XtNewString(fileName);
   }

else if (convertInfo->reason == DtCR_DND_CONVERT_DELETE)
   {
  deleteFile(fileName);
  } else {
  convertInfo->status = DtDND_FAILURE;
  }

}
```

ドロップ領域の使い方

この節では、ドロップ領域の使い方を説明します。

ドロップ領域の登録

一般に、ドロップ領域は、ドロップ領域になるウィジェットが作成された直後に登録します。モード付きドロップ領域にする場合は、ユーザがその上にドロップできるようにするにはウィジェットをドロップ領域として登録し、ユーザがその上にドロップできないようにするには登録を解除します。

Motif テキスト・ウィジェットは、作成されたときに、テキスト用のドロップ領域として自動的に登録されます。二重登録が可能です。テキスト・ウィジェットが、テキストだけでなく、ファイル名など他のデータのドロップも受け入れるようにする場合には、テキスト・ウィジェットをファイル用のドロップ領域としても登録できます。Motif によって提供されるテキスト・ドロップ機能は変わりません。ファイル名 (または他のデータ型) のドロップに対する機能は、その上に重ねられます。

ウィジェットをドロップ領域として登録するには、関数 `DtDndDropRegister()` を使用します。この関数は、必要に応じて二重登録を処理し、デスクトップに特有の

準備を行い、XmDropSiteRegister() を呼び出します。DtDndDropRegister() 関数の形式とパラメータの使用法は、次のとおりです。

```
void
DtDndDropRegister(
  Widget  dropSite,
 DtDndProtocol protocols;
 unsigned char operations;
  XtCallbackList transferCallback;
 ArgList argList;
  Cardinal argCount)
```

Widget dropSite

ドロップ領域として登録されるウィジェット

DtDndProtocol protocols

ドロップ領域が使用できるデータ転送プロトコルのリストを指定します。複数のプロトコルの使用を指定するには、or (|) とプロトコルの値を使用します。

unsigned char operations

ドロップ領域によってサポートされる操作。ドロップ領域は、目的の操作の組み合わせに対して or (|) を使用することによって、XmDROP_MOVE、XmDROP_COPY、および XmDROP_LINK の任意の組み合わせをサポートできます。

XtCallbackList transferCallback

この関数は、ドロップ領域にドロップされたデータを受け入れます。転送コールバックについては、次の節で詳しく説明します。

ArgList argList

オプションの引き数リストを指定します。

Cardinal argCount

argList 内の引き数の数を指定します。

転送コールバックの使い方

転送コールバックは、ドロップが発生したときに、ドラッグ・ソースからデータを受け入れます。転送コールバックの最初のアクションは、callData の中の reason フィールドの確認です。reason が DtCR_DND_TRANSFER_DATA ではない場合には、直ちに戻さなければなりません。そうでない場合には、型と reason の中で指定された操作に基づいて、データ転送を続けます。たとえば、ファイルのコピーを処理している場合には、データ構造体からファイル名を検索し、ファイルを開き、その内容をコピーします。ドロップ領域が複数のデータ型をサポートしている場合には、各データ型の転送を適切にサポートする必要があります。

次に、テキストとファイル名のデータ型のコピーをサポートするドロップ領域を描画するための簡単な転送コールバックを示します。

```
void
 TransferCallback(
  Widget widget,
  XtPointer clientData,
  XtPointer callData)
{
   DtDndTransferCallbackStruct *transferInfo =
    (DtDndTransferCallbackStruct*) callData;
   int ii;

   DtDndcontext * dropData = transferInfo->dropData;
    return;
   switch dropData->protocol {
  case DtDND_FILENAME_TRANSFER:
   for (ii=0; ii < dropData->numItems; ii++) {
    drawTheString(dropData->data, strings[ii]);
   }
   break;
  case DtDND_TEXT_TRANSFER:
   for (ii=0; ii<dropData->numItems; ii++){
     drawTheFile(dropData->data.files[ii]);
   }
     break;
  default:
   transferInfo->status = DtDND_FAILURE;
     }
}
```

データ型の使い方

バッファのドロップを受け入れるアプリケーションでは、ドロップされたデータをその型に基づいて異なる方法で処理したいことがあります。データ型を判断するには、データ型 API を使用します。重要なデータ型関数呼び出しは、DtDtsBufferToDataType() と DtDtsBufferToAttributeValue() です。

前者はデータのデータ属性名を返し、後者は指定されたデータ属性の値を返します。ドラッグ & ドロップに役立つ属性を、表 5–8 に示します。

表 5–8 データ型属性

属性	説明
ICON	このデータに対して使用するアイコンのパス
MEDIA	このデータに対するメッセージ提携メディア名

詳細は、第 9 章を参照してください。

パート III　オプションの統合方法

第 6 章から第 10 章では、次のオプションの統合方法の作業を実行する方法について説明します。

- ワークスペース・マネージャと統合し、アプリケーションがセッションの開始時に各セッションのワークスペースの位置を判断できるようにする
- CDE カスタム・ウィジェットを使用する
- アプリケーションの内部からアクションを呼び出す
- データ型データベースにアクセスする
- カレンダの API にアクセスする
- 第 6 章「ワークスペース・マネージャとの統合」
- 第 7 章「共通デスクトップ環境の Motif ウィジェット」
- 第 8 章「アプリケーションからのアクションの実行」
- 第 9 章「データ型データベースのアクセス」
- 第 10 章「カレンダとの統合」

第 6 章

ワークスペース・マネージャとの統合

ワークスペース・マネージャは、デスクトップの複数のワークスペース環境の中でアプリケーションがウィンドウを管理するための手段を提供します。アプリケーションは、ワークスペース・マネージャと通信することによって、主に次の 4 つのタスクを実行できます。

- アプリケーションのウィンドウを 1 つ以上のワークスペースに置く
- アプリケーションのウィンドウが位置するワークスペースを識別する
- アプリケーションのウィンドウが別のワークスペースに移動するのを防止する
- ユーザが別のワークスペースに切り替えるなど、ワークスペースに対する変更を監視する

通常、セッション・マネージャはプログラマに干渉されることなく、アプリケーションのメイン・ウィンドウを正しいワークスペースに取り出します。ただし、アプリケーションが複数のトップレベル・ウィンドウを持つ場合には、ワークスペース・マネージャの API を使用してウィンドウの位置を特定し、このデータをセッション状態の一部として保存しなければなりません。

セッション間のアプリケーション関連情報の保存の詳細は、第 4 章を参照してください。

- 84ページの「ワークスペース・マネージャとの通信」
- 85ページの「アプリケーション・ウィンドウをワークスペースに置く」
- 86ページの「アプリケーション・ウィンドウがあるワークスペースの識別」
- 87ページの「ワークスペース間のアプリケーションの移動防止」
- 88ページの「ワークスペースの変更の監視」

ワークスペース・マネージャとの通信

アプリケーションは、デスクトップが提供する関数を使用して、ワークスペース・マネージャと通信します。これらの関数を使用すると、ワークスペースの管理に関連するさまざまなタスクをすばやく簡単に実行できます。次に、これらの関数のリストを示します。

- `DtWsmAddCurrentWorkspaceCallback()`
- `DtWsmAddWorkspaceFunctions ()`
- `DtWsmAddWorkspaceModifiedCallback ()`
- `DtWsmFreeWorkspaceInfo ()`
- `DtWsmGetCurrentBackdropWindows ()`
- `DtWsmGetCurrentWorkspace ()`
- `DtWsmGetWorkspaceInfo ()`
- `DtWsmGetWorkspaceList ()`
- `DtWsmGetWorkspacesOccupied ()`
- `DtWsmOccupyAllWorkspaces ()`
- `DtWsmRemoveWorkspaceCallback ()`
- `DtWsmRemoveWorkspaceFunctions ()`
- `DtWsmSetCurrentWorkspace ()`
- `DtWsmSetWorkspacesOccupied()`

2 つのデモ・プログラム (`occupy.c` と `wsinfo.c`) の中に、これらの関数の使用方法を示すコードの一部分があります。`occupy.c`、`wsinfo.c`、およびいくつかのブランドのワークステーションの makefile をリストしたものがディレクトリ `/usr/dt/examples/dtwsm` にあります。各関数の詳細は、該当するマニュアル・ページを参照してください。

アプリケーション・ウィンドウをワークスペースに置く

アプリケーションは、ウィンドウを任意のまたは全部の既存のワークスペースに置くことができます。DtWsmOccupyAllWorkspaces() は、現在定義されているすべてのワークスペースにウィンドウを置きます。一方、DtWsmSetWorkspacesOccupied() は、この関数に渡されたリストの中で指定されている、すべてのワークスペースにウィンドウを置きます。

▼ アプリケーション・ウィンドウをすべてのワークスペースに置くには

♦ DtWsmOccupyAllWorkspaces() を使用します。

occupy.c では、[すべてのワークスペースに配置] プッシュ・ボタンのコールバック allWsCB() がこの関数を呼び出します。

```
DtWsmOccupyAllWorkspaces (XtDisplay(toplevel),
                          XtWindow(toplevel));
```

- XtDisplay(toplevel) は、X ディスプレイです。
- XtWindow(toplevel) は、すべてのワークスペースに置かれるウィンドウです。

この関数の詳細は、DtWsmOccupyAllWorkspaces() のマニュアル・ページを参照してください。

▼ アプリケーション・ウィンドウを指定されたワークスペースに置くには

♦ DtWsmSetWorkspacesOccupied() を使用します。

occupy.c では、[配置するワークスペース] プッシュ・ボタンのコールバック setCB() がこの関数を呼び出します。

```
DtWsmSetWorkSpacesOccupied XtDisplay(toplevel),
                           XtWindow(toplevel), paWsSet, numSet);
```

- XtDisplay(toplevel) は、X ディスプレイです。
- XtWindow(toplevel) は、ワークスペースに置かれるウィンドウです。

- paWsSet は、X のアトムに変換されたワークスペース名のリストを指すポインタです。
- numSet は、リスト内のワークスペースの番号です。

この関数の詳細は、DtWsmSetWorkspacesOccupied() のマニュアル・ページを参照してください。

アプリケーション・ウィンドウがあるワークスペースの識別

関数 DtWsmGetWorkspacesOccupied() は、指定されたアプリケーション・ウィンドウがあるワークスペースのリストを返します。occupy.c では、プロシージャ ShowWorkspaceOccupancy() がこの関数を呼び出します。この呼び出しの結果に基づいて、ShowWorkspaceOccupancy() はワークスペースを表すトグル・ボタンの外観を変更します。アプリケーション・ウィンドウがあるワークスペースの各トグル・ボタンには、チェック・マークが表示されます。

▼ アプリケーション・ウィンドウがあるワークスペースを識別するには

♦ DtWsmGetWorkspacesOccupied() を使用します。

```
rval = DtWsmGetWorkspacesOccupied(XtDisplay(toplevel)
           XtWindow(toplevel), &paWsIn, &numWsIn);
```

- XtDisplay(toplevel) は、X ディスプレイです。
- XtWindow(toplevel) は、ワークスペースで検索されるウィンドウです。
- paWsIn は、X のアトムに変換されたワークスペース名のリストを指すポインタのアドレスです。
- numWsIn は、リスト内のワークスペースの番号を表す整数のアドレスです。

この呼び出しの後、ループが設定され、ワークスペースのリスト (DtWsmGetWorkspaceList() によってプロシージャ SetUpWorkspaceButtons() で検索されます) と、アプリケーション・ウィンドウがあることがわかったワークスペースのリストとを比較します。各トグル・ボタン

がアプリケーション・ウィンドウがあるワークスペースを表す場合は、トグル・ボタン・リソース XmNset に True が設定されます。

ワークスペース間のアプリケーションの移動防止

関数 DtWsmRemoveWorkspaceFunctions() は、アプリケーションが次の作業を実行しないようにします。

- 他のワークスペースへの切り替え
- すべてのワークスペースの占有
- 現在のワークスペースからの削除

DtWsmRemoveWorkspaceFunctions() が上記の作業を実行する場合は、デスクトップ・ワークスペース・マネージャ (dtwm) のウィンドウ・メニューの一部分をアクティブにしないようにします。dtwm はアプリケーションのトップレベル・ウィンドウを管理するときにワークスペース情報をチェックするだけなので、アプリケーションはトップレベル・ウィンドウがマップされる前に DtWsmRemoveWorkspaceFunctions() を呼び出さなければなりません。アプリケーションのトップレベル・ウィンドウが管理された後で DtWsmRemoveWorkspaceFunctions() を呼び出す必要がある場合には、最初に Xlib 関数 XWithdrawWindow() を呼び出してから DtWsmRemoveWorkspaceFunctions() を呼び出し、次に XMapWindow() を呼び出して、トップレベル・ウィンドウを再マップしなければなりません。

▼ 別のワークスペースへの移動を防止するには

◆ DtWsmRemoveWorkspaceFunctions() を使用します。

```
DtWsmRemoveWorkspaceFunctions(XtDisplay(toplevel),
                              XtWindow(toplevel));
```

- XtDisplay(toplevel) は、X ディスプレイです。

- XtWindow(toplevel) は、ワークスペースの移動を防止するウィンドウです。

ワークスペースの変更の監視

次の関数の 1 つまたは両方を使用して、ワークスペースの変更を監視できます。

- DtWsmAddCurrentWorkspaceCallback()
- DtWsmWorkspaceModifiedCallback()

DtWsmAddCurrentWorkspaceCallback() は、ワークスペース・マネージャが新しいワークスペースに切り替えられるときに、必ず呼び出されるアプリケーション・コールバックを登録します。詳細は、DtWsmAddCurrentWorkspaceCallback(3) のマニュアル・ページを参照してください。

DtWsmWorkspaceModifiedCallback() は、ワークスペースが追加、削除、または変更されるときに必ず呼び出されるアプリケーション・コールバックを登録します。詳細は、DtWsmWorkspaceModifiedCallback(3) のマニュアル・ページを参照してください。

▼ ワークスペースの切り替えを監視するには

♦ DtWsmAddCurrentWorkspaceCallback() を使用します。

デモ・プログラム wsinfo.c では、この関数は、トップレベル・ウィジェットが実体化された後で呼び出されます。

```
DtWsmAddCurrentWorkspaceCallback (toplevel, wschangecb, NULL);
```

- toplevel は、アプリケーションのトップレベル・ウィジェットです。
- wschangecb() は、呼び出される関数名です。
- NULL は、コールバックに渡されるクライアント・データのパラメータです。この場合、データは渡されません。

▼ 他のワークスペースの変更を監視するには

♦ DtWsmWorkspaceModifiedCallback() を使用します。

```
DtWsmWorkspaceModifiedCallback toplevel, wschangecb, NULL);
```

- toplevel は、アプリケーションのトップレベル・ウィジェットです。
- wschangecb() は、呼び出される関数名です。
- NULL は、コールバックに渡されるクライアント・データのパラメータです。この場合、データは渡されません。

第 7 章

共通デスクトップ環境の Motif ウィジェット

共通デスクトップ環境 (CDE) は、Motif 2.1 ライブラリ (バグ修正付き) および拡張機能を提供します。さらに、CDE は、OPEN LOOK と Microsoft Windows の特定の機能を提供するために使用できる 4 つのカスタム・ウィジェットを提供します。この章では、これらの Motif カスタム・ウィジェットについて説明します。

- 92ページの「メニュー・ボタン・ウィジェット (DtMenuButton)」
- 97ページの「テキスト・エディタ・ウィジェット (DtEditor)」

ウィジェット・ライブラリ (libDtWidget) には、既存の Motif 2.1 ウィジェットの機能を組み合わせたり、拡張したりする 4 つのウィジェットがあります。

- DtMenuButton は、メニュー・バーの外側にメニュー階層機能を提供します。
- DtEditor は、カット & ペーストなどの単純なテキスト・エディタ関数を組み込みます。
- DtSpinBox は、テキスト・フィールドと矢印ボタンを組み合わせたコントロールを作成して、数値またはテキスト値を増減できます。このウィジェットは Motif ウィジェットの XmSpinBox を引き継いでいます。
- DtComboBox は、テキスト・フィールドとリスト・ボックスを組み合わせたコントロールを作成して、テキスト・フィールドに対する多数の有効な選択項目の 1 つを表示します。このウィジェットは Motif ウィジェットの XmComboBox を引き継いでいます。

これらのウィジェットは、すべての CDE アプリケーションに共通の機能を提供します。これらのウィジェットは、サブクラス化はサポートしていません。

カスタム・ウィジェット・ライブラリは、次のライブラリに直接依存します。

- Motif スーパークラスのサポートについては Xm ライブラリ

- ウィジェットの作成と操作については Xt ライブラリ
- ベース X Window System については X11 ライブラリ
- DtEditor が利用するデスクトップ・サポートについては DtSvc

メニュー・ボタン・ウィジェット (DtMenuButton)

DtMenuButton ウィジェットは、メニュー区画の外側にメニュー階層機能を提供するために使用します。

DtMenuButton ウィジェットは、XmCascadeButton ウィジェットのメニュー階層機能を補足するコマンド・ウィジェットです。XmCascadeButton ウィジェットを補うものとして、メニュー・バー、プルダウン、またはポップアップの外側で示すことができます (MenuPane の内部では XmCascadeButton ウィジェットを使用します)。図 7–1 に、DtMenuButton ウィジェットの使用例を示します。

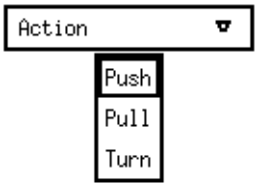

図 7–1　メニュー・ボタン・ウィジェット (DtMenuButton) の例

ライブラリとヘッダ・ファイル

DtMenuButton ウィジェットは、libDtWidget ライブラリにあります。ヘッダ・ファイルは、Dt/MenuButton.h です。

デモ・プログラム

DtMenuButton ウィジェットの使用例が入っているデモが、/usr/dt/examples/dtwidget/controls.c にあります。

簡易関数

DtCreateMenuButton() は、CDE ウィジェットを作成する簡易関数です。

DtMenuButton ウィジェットは、XmLabel クラスのサブクラスです。視覚的には、DtMenuButton ウィジェットは、ラベル文字列とメニュー・グリフ (絵文字) を持ちます。メニュー・グリフは、常にウィジェットの右端に表示され、デフォルトでは下向き矢印です。

DtMenuButton ウィジェットには、暗黙的に作成されたサブメニューが接続されています。サブメニューは、この DtMenuButton ウィジェットを親とするポップアップ・メニューです。暗黙的に作成されたサブメニュー名は、この DtMenuButton ウィジェットの名前の前に submenu_ を付けたものです。サブメニューのウィジェット ID は、この DtMenuButton ウィジェットの DtNsubMenuId リソースに XtGetValues を設定することにより取得できます。暗黙的に作成されたサブメニューは、このウィジェットの利用者によって破壊されることはありません。

サブメニューは、DtMenuButton ウィジェットのどこかで [メニュー・ポスト] ボタン(XmRowColumn の XmNmenuPost リソースを参照) を押すことによって、または Motif の取り消しキー (通常は [Escape] キー) を押すことによってポップアップできます。

クラス

DtMenuButtonWidget は、Core、XmPrimitive、および XmLabel クラスから動作とリソースを継承します。

クラス・ポインタは、dtMenuButtonWidgetClass です。

クラス名は、DtMenuButtonWidget です。

DtMenuButtonWidget は、サブクラス化をサポートしません。

リソース

DtMenuButtonWidget は、次のリソースを提供します。これらのリソースのクラス、型、デフォルト、およびアクセスを表 7–1 に示します。

- DtNcascadingCallback は、接続されたサブメニューが表示される前に呼び出されるコールバックのリストを指定します。

- DtNcascadePixmap は、メニュー・グリフとして表示されるピックスマップを指定します。ピックスマップが指定されない場合は、下向き矢印が表示されます。
- DtNsubMenuId は、この DtMenuButton ウィジェットと関連付けられるポップアップ・メニュー区画のウィジェット ID を指定します。この DtMenuButton を親としてポップアップ・メニュー区画を作成しなければなりません。リソースの設定時に、このウィジェットによってサブメニューが自動的に破壊されるので、ウィジェットの作成時にこのリソースを指定できません。

詳細は、DtMenuButtonwidget(3X) のマニュアル・ページを参照してください。

アクセス欄のコードは、次の作業が可能かどうかを示します。

- 作成時にリソースを設定する (C)
- XtSetValues を使用して設定する (S)
- XtGetValues を使用して検索する(G)

表 7–1 DtMenuButtonWidget リソース

名前	クラス	型	デフォルト	アクセス
DtNcascadingCallback	DtCCallback	XtCallbackList	NULL	C
DtNcascadePixmap	DtCPixmap	Pixmap	XmUNSPECIFIED_PIXMAP	CSG
DtNsubMenuId	DtCMenuWidget	Widget	NULL	SG

コールバックのための構造体

コールバックのための構造体を次に示し、表 7–2 で説明します。

```
typedef struct {
  int   reason;
  XEvent *event;
} XmAnyCallbackStruct;
```

表 7–2 DtMenuButtonWidget コールバックのための構造体

構造体	説明
reason	コールバックが呼び出された reason を返します。
event	コールバックをトリガした XEvent へのポインタ。コールバックが XEvent によってトリガされなかった場合には NULL になります。

DtMenuButton ウィジェットの例

次の例は、DtMenuButton ウィジェットの作成方法と使用方法を示しています。このコードは、/usr/dt/examples/dtwidget ディレクトリの controls.c デモの一部です。

```
/*
  * Example code for DtMenuButton
  */

#include Dt/DtMenuButton.h

/* MenuButton custom glyph */

#define menu_glyph_width 16
#define menu_glyph_height 16
static unsigned char menu_glyph_bits[] = {
  0xe0, 0x03, 0x98, 0x0f, 0x84, 0x1f, 0x82, 0x3f, 0x82, 0x3f, 0x81,
  0x7f, 0x81, 0x7f, 0xff, 0x7f, 0xff, 0x40, 0xff, 0x40, 0xfe, 0x20, 0xfe,
  0x20, 0xfc, 0x10, 0xf8, 0x0c, 0xe0, 0x03, 0x00, 0x00};

static void CreateMenuButtons(Widget parent) {
     Widget menuButton, submenu, titleLabel, button;
     Pixmap cascadePixmap;
     Pixel fg, bg;
     Cardinal depth;
     XmString labelString;
     Arg args[20];
     int i, n;

      /* Create title label */

     labelString = XmStringCreateLocalized(``MenuButton Widget'');
     n = 0;
     XtSetArg(args[n], XmNlabelString, labelString); n++;
     titleLabel = XmCreateLabel(parent, ``title'', args, n);
     XtManageChild(titleLabel);
     XmStringFree(labelString);

     /*
      * Create a MenuButton.
      * Add push buttons to the built-in popup menu.
```

```
     */

     labelString = XmStringCreateLocalized(''Action''); n = 0;
     XtSetArg(args[n], XmNlabelString, labelString); n++;
     menuButton = DtCreateMenuButton(parent, ''menuButton1'', args, n);
     XtManageChild(menuButton);
     XmStringFree(labelString);
     XtVaGetValues(menuButton, DtNsubMenuId, &submenu, NULL);
     button = XmCreatePushButton(submenu, ''Push'', NULL, 0);
     XtManageChild(button);
     button = XmCreatePushButton(submenu, ''Pull'', NULL, 0);
     XtManageChild(button);
     button = XmCreatePushButton(submenu, ''Turn'', NULL, 0);
     XtManageChild(button);

     /*
      * Create a MenuButton.
      * Replace the built-in popup menu with a tear-off menu.
      * Add a custom pixmap in the colors of the MenuButton.
      */

     labelString = XmStringCreateLocalized(''Movement'');
     n = 0;
     XtSetArg(args[n], XmNlabelString, labelString); n++;
     menuButton = DtCreateMenuButton(parent, ''menuButton1'', args, n);
     XtManageChild(menuButton);
     XmStringFree(labelString);

     /* Create a tear-off menu */

     n = 0;
     XtSetArg(args[0], XmNtearOffModel, XmTEAR_OFF_ENABLED); n++;
     submenu = XmCreatePopupMenu(menuButton, ''submenu'', args, n);
     button = XmCreatePushButton(submenu, ''Run'', NULL, 0);
     XtManageChild(button);
     button = XmCreatePushButton(submenu, ''Jump'', NULL, 0);
     XtManageChild(button);
     button = XmCreatePushButton(submenu, ''Stop'', NULL, 0);
     XtManageChild(button);

     XtVaSetValues(menuButton, DtNsubMenuId, submenu, NULL);

     /* Create a pixmap using the menu button's colors and depth */

     XtVaGetValues(menuButton,
    XmNforeground, &fg,
    XmNbackground, &bg,
    XmNdepth, &depth,
    NULL);

     cascadePixmap = XCreatePixmapFromBitmapData(XtDisplay
    (menuButton),DefaultRootWindow(XtDisplay
    (menuButton)),
    (char*)menu_glyph_bits,
    menu_glyph_width, menu_glyph_height,
     fg, bg, depth);
     XtVaSetValues(menuButton, DtNcascadePixmap, cascadePixmap,
    NULL);
}
```

テキスト・エディタ・ウィジェット (DtEditor)

共通デスクトップ環境のテキスト編集システムは、次の 2 つのコンポーネントから成ります。

- グラフィカル・インタフェース、アクション・インタフェース、および ToolTalk インタフェースを介して編集サービスを提供するテキスト・エディタ・クライアントの `dtpad`
- 次の編集サービスのためのプログラム・インタフェースを提供するエディタ・ウィジェットの `DtEditor(3)`
  - カット & ペースト
  - 検索と置換
  - 単純な書式化
  - スペルチェック (8 ビット・ロケール用)
  - 以前の編集を元に戻す
  - ASCII テキスト、マルチバイト・テキスト、およびバッファ・データの入出力をサポートする拡張入出力処理機能
  - ファイルの直接読み書きのサポート

OSF/Motif テキスト・ウィジェットはプログラム・インタフェースも提供しますが、システム全体で一貫したエディタを使用するアプリケーションは、`DtEditor(3)` ウィジェットを使用しなければなりません。CDE のテキスト・エディタとメール・プログラムは、エディタ・ウィジェットを使用します。このウィジェットは、次のような状況のときに使用してください。

1. **[スペルチェック]、[元に戻す]、および [検索／変更] など、`DtEditor(3)` ウィジェットが提供する機能を使いたい場合**
2. **ユーザがファイルからデータを読み込んだり、ファイルにデータを書き込んだりするコードを作成したくない場合**
3. **ユーザが入力した文字またはユーザが行なったカーソル移動を調べる必要がないプログラムを作成する場合**

この節では、テキスト・エディタ・ウィジェット `DtEditor(3)` について説明します。

エディタ・ウィジェット・ライブラリは、テキスト・ファイルの作成と編集のためのサポートを提供します。デスクトップ環境で実行するアプリケーションで、一貫した方法でテキスト・データを編集できるようにします。DtEditor(3) ウィジェットは、テキスト用のスクロールする編集ウィンドウ、オプションのステータス行と、テキストの検索と置換、スペルチェック、および書式オプションの指定を行うためのダイアログから成ります。テキスト・エディタ・ウィジェットには、ウィジェットをプログラム的に制御するための簡易関数のセットが含まれています。

ライブラリとヘッダ・ファイル

DtEditor ウィジェットは、libDtWidget ライブラリにあります。ヘッダ・ファイルは、Dt/Editor.h です。

デモ・プログラム

DtEditor ウィジェットの使用例が入っているデモが、/usr/dt/examples/dtwidget/editor.c にあります。

クラス

DtEditor ウィジェット・クラスについては、ウィジェットのサブクラス化はサポートされません。

DtEditor は、Core、Composite、Constraints、XmManager、および XmForm クラスから動作とリソースを継承します。

エディタ・ウィジェットのクラス名は、DtEditorWidget です。

クラス・ポインタは、dtEditorWidgetClass です。

簡易関数

DtEditor 簡易関数を、次の表に示します。

ライフ・サイクル関数

DtEditor ライフ・サイクル関数を表 7–3 に示します。

表 7–3　DtEditor ライフ・サイクル関数

関数	説明
DtCreateEditor	DtEditor ウィジェットの新規インスタンスとその子を作成します。
DtEditorReset	DtEditor ウィジェットを初期状態に復元します。

入出力関数

DtEditor 入出力関数を表 7–4 に示します。

表 7–4　DtEditor 入出力関数

関数	説明
DtEditorAppend	エディタ・ウィジェットの最後に内容データを追加します。
DtEditorAppendFromFile	エディタ・ウィジェットの最後にファイルの内容を追加します。
DtEditorGetContents	エディタ・ウィジェットの内容全体を検索します。
DtEditorInsert	内容データを現在の挿入位置に挿入します。
DtEditorInsertFromFile	ファイルの内容を現在の挿入位置に挿入します。
DtEditorReplace	テキストの一部を与えられたデータと置き換えます。
DtEditorReplaceFromFile	テキストの一部をファイルの内容と置き換えます。
DtEditorSaveContentsToFile	現在選択されている内容を空白に置き換えます。

表 7–4　DtEditor 入出力関数　続く

関数	説明
DtEditorSetContents	内容データをエディタ・ウィジェットに読み込んで、ウィジェットの内容全体を置き換えます。
DtEditorSetContentsFromFile	ファイルの内容をエディタ・ウィジェットに読み込んで、ウィジェットの内容全体を置き換えます。

選択関数

DtEditor 選択関数を表 7–5 に示します。

表 7–5　DtEditor 選択関数

関数	説明
DtEditorClearSelection	現在選択されている内容を空白に置き換えます。
DtEditorCopyToClipboard	現在選択されている内容をクリップボードにコピーします。
DtEditorCutToClipboard	現在選択されている内容を削除して、クリップボードに入れます。
DtEditorDeleteSelection	現在選択されている内容を削除します。
DtEditorDeselect	選択されている内容を選択解除します。
DtEditorPasteFromClipboard	クリップボードの内容をエディタ・ウィジェットにペーストして、現在選択されている内容を置き換えます。
DtEditorSelectAll	エディタ・ウィジェットの内容全体を選択します。

書式化関数

DtEditor 書式化関数を表 7–6 に示します。

表 7–6 DtEditor 書式化関数

関数	説明
DtEditorFormat	エディタ・ウィジェットの内容の全部または一部を書式化します。
DtEditorInvokeFormatDialog	[書式] ダイアログ・ボックスを表示して、マージンと位置揃えのスタイルに関する書式設定を指定し、書式操作を実行できます。

検索／変更関数

DtEditor検索／変更関数を表 7–7 に示します。

表 7–7 DtEditArea 検索／変更関数

関数	説明
DtEditorChange	文字列の 1 つまたはすべての存在箇所を置換します。
DtEditorFind	文字列の次の出現箇所を検索します。
DtEditorInvokeFindChangeDialog	文字列を検索 (オプションで置換も) するためのダイアログ・ボックスを表示します。
DtEditorInvokeSpellDialog	現在の内容の中でスペルが間違っている単語のリストがあるダイアログ・ボックスを表示します。

補助関数

DtEditor 補助関数を表 7–8 に示します。

表 7–8 DtEditor 補助関数

関数	説明
DtEditorCheckForUnsavedChanges	エディタ・ウィジェットの内容が前回の検索または保存以後に変更されているかどうかを報告します。
DtEditorDisableRedisplay	ビジュアル属性が変更された場合でも、エディタ・ウィジェットの再表示をしません。
DtEditorEnableRedisplay	エディタ・ウィジェットの表示の更新を強制します。
DtEditorGetInsertPosition	エディタ・ウィジェットの挿入カーソル位置を返します。
DtEditorGetLastPosition	編集ウィンドウの最後の文字の位置を返します。
DtEditorGetMessageTextFieldID	アプリケーション・メッセージを表示するために使用されるテキスト・フィールド・ウィジェットのウィジェット ID を検索します。
DtEditorGetSizeHints	エディタ・ウィジェットからサイズ情報を検索します。
DtEditorGoToLine	挿入カーソルを指定された行へ移動します。
DtEditorSetInsertionPosition	挿入カーソルの位置を設定します。
DtEditorTraverseToEditor	エディタ・ウィジェットの編集ウィンドウへのキーボード移動を設定します。
DtEditorUndoEdit	ユーザが行なった最後の編集を元に戻します。

リソース

DtEditor ウィジェットは、次のリソースのセットを提供します。

- DtNautoShowCursorPosition に True が設定された場合は、スクロール編集ウィンドウに表示されるテキストに挿入カーソルが必ずあるようにします。挿

入カーソルが変わると、エディタの内容をスクロールして、挿入ポイントがウィンドウに入るようにします。

- DtNblinkRate は、テキスト・カーソルの点滅間隔をミリ秒単位で指定します。挿入カーソルの点滅に要する時間は、点滅間隔の 2 倍です。点滅間隔が 0 に設定された場合は、カーソルは点滅しません。値を負にすることはできません。
- DtNbuttonFontList は、DtEditor のダイアログ・ボックスに表示されるボタンのフォント・リストを指定します。
- DtNcolumns は、エディタの初期の幅を整数の文字単位で指定します。値は 0 より大きくなければなりません。
- DtNcursorPosition は、エディタの中で現在の挿入カーソルが置かれる位置を指定します。位置は、テキストの先頭からの文字数によって決められます。最初の文字位置は 0 です。
- DtNcursorPositionVisible は、論理値が True のときに点滅しているテキスト・カーソルで、挿入カーソルの位置をマークします。
- DtNdialogTitle は、DtEditor によって表示されるすべてのダイアログのタイトルを指定します。これには、単語の検索と置換、スペルが間違っている単語、および書式設定のためのダイアログが含まれます。
- DtNeditable に True が設定されている場合は、ユーザはデータを編集できます。False が設定されている場合は、ユーザはデータを編集できません。
- DtNlabelFontList は、DtEditor ラベルとして使用されるフォント・リストを指定します (ラベルは、ステータス行と DtEditor ダイアログ・ボックスに表示されます)。
- DtNoverstrike に False が設定されている場合、エディタ・ウィジェットに入力された文字は、カーソルの位置に挿入されます (デフォルト)。True が設定されている場合、エディタ・ウィジェットに入力された文字は、挿入カーソルの直後の文字を置き換えます。行末に達した場合は、文字は行末に追加されます。ステータス行が表示されている場合、DtNoverstrike が True のときは DtNoverstrikeIndicatorLabel が必ずステータス行に表示されます。
- DtNrows は、エディタの初期の高さを文字単位で指定します。値は 0 より大きくなければなりません。
- DtNscrollHorizontal は論理値が True の場合、テキストを水平方向にスクロールできるスクロール・バーを追加します。
- DtNscrollLeftSide は論理値が True の場合は、スクロール編集ウィンドウの左側に垂直スクロール・バーを置きます。

- DtNshowStatusLine は、True が設定された場合、テキスト・ウィンドウの下にステータス行を表示します。ステータス行のフィールドには、挿入カーソルの現在の行番号、ドキュメント内の総行数、およびエディタが上書きモードかどうかを表示します。ユーザは、行番号表示に行番号を入力することによって、直接その行を表示できます。

ステータス行は、アプリケーションによって提供されるメッセージを表示するための Motif の XmTextField(3x) ウィジェットも含んでいます。このフィールドは、アプリケーションが編集中のドキュメントについてのステータスとフィードバックを表示するのに便利です。テキスト・フィールドの ID は、DtEditorGetMessageTextFieldID(3) を使用して検索されます。メッセージは、このウィジェットの XmNvalue または XmNvalueWcs リソースを設定することによって表示されます。テキスト・フィールドが必要ない場合には、その ID で XtUnmanageWidget(3X) を呼び出すことによって管理からはずことができます。

- DtNspellFilter は、スペルチェックに使用するフィルタを指定します。関数 DtEditorInvokeSpellDialog(3) は、DtNspellFilter によって指定されたフィルタを使用してエディタの内容をチェックします。指定されたフィルタは、ファイル名を受け入れて、このファイルの中のスペルが間違っている単語と認識不能の単語のリストを標準出力に出力しなければなりません。デフォルトのフィルタは、spell(1) です。

- DtNtextBackground は、編集ウィンドウのバックグラウンドを指定します。

- DtNtextDeselectCallback は、編集領域内でテキストが選択されていない場合に呼び出される関数を指定します。コールバックによって送られる reason は、DtEDITOR_TEXT_DESELECT です。

- DtNtextFontList は、DtEditor 編集ウィンドウとテキスト・フィールドに使用されるフォント・リストを指定します。テキスト・フィールドは、ステータス行と DtEditor ダイアログ・ボックスに表示されます。

- DtNtextForeground は、編集ウィンドウのフォアグラウンドを指定します。

- DtNtextSelectcallback は、編集領域内でテキストが選択された場合に呼び出される関数を指定します。コールバックによって送られる reason は、DtEDITOR_TEXT_SELECT です。

- DtNtextTranslations は、編集ウィンドウに追加される変換を指定します。このリソースで指定された変換は、編集ウィンドウに対して定義された重複する変換を無効にします。DtEditor によって提供される変換のリストについては、DtEditor(3) のマニュアル・ページを参照してください。

- DtNtopCharacter は、スクロールした編集ウィンドウの最上部にテキストの位置がある行を表示します。行はテキストを左右に移動することなくウィジェットの最上部に表示されます。位置は、テキストの先頭からの文字数によって決められます。最初の文字位置は 0 です。

  DtNtopCharacter に対する XGetValues(3X) は、ウィジェットの最上部に表示される行の最初の文字の位置を返します。

- DtNwordWrap は、ウィンドウの右端に達した場合に、語の切れ目でソフト・キャリッジ・リターンによる行分割を行います。行折返しは、エディタ・ウィジェットの内容の視覚的外観だけに影響を及ぼすので注意してください。行分割(ソフト・キャリッジ・リターン) は、テキストに物理的に挿入されるわけではありません。エディタは、ウィジェットの内容が検索される場合またはファイルに保存される場合に、ハード・キャリッジ・リターンへの置換をサポートします。詳細は、DtEditorGetContents(3) および DtEditorSaveContentsToFile(3) のマニュアル・ページを参照してください。

各リソースのクラス、型、デフォルト、およびアクセスを表 7–9 にリストします。継承クラスのリソース値を設定することによって、このウィジェットの属性を設定することもできます。.Xdefaults ファイルの中で名前またはクラスによってリソースを参照するには、DtN または DtC の接頭辞を除いて、残りの文字を使用します。.Xdefaults ファイルでリソースに対して定義済みの値の 1 つを指定するには、Dt 接頭辞を除いて、残りの文字を使用します (小文字または大文字で、単語の間に下線を入れます)。

アクセス欄のコードは、次の作業が可能かどうかを示します。

- 作成時にリソースを設定する (C)
- XtSetvalues を使用して設定する (S)
- XtGetValues を使用して検索する (G)

詳細は、DtEditor(3) のマニュアル・ページを参照してください。

表 7–9 DtEditor リソース

名前	クラス	型	デフォルト	アクセス
DtNautoShowCursorPosition	DtCAutoShowCursorPosition	Boolean	True	CSG
DtNblinkRate	DtCBlinkRate	int	500	CSG

表 7–9　DtEditor リソース　続く

名前	クラス	型	デフォルト	アクセス
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
DtNshowStatusLine	DtCShowStatusLine	Boolean	False	CSG
DtNspellFilter	DtCspellFilter	char *	Spell	CSG
DtNtextBackground	DtCBackground	Pixel	Dynamic	CSG
DtNtextDeselectCallback	DtCCallback	XtCallbackList	NULL	C

表 7–9　DtEditor リソース　続く

名前	クラス	型	デフォルト	アクセス
DtNtextFontList	DtCFontList	XmFontList	Dynamic	CSG
DtNtextForeground	DtCForeground	Pixel	Dynamic	CSG
DtNtextTranslations	DtCTranslations	XtTranslations	NULL	CS
DtNtextSelectCallback	DtCCallback	XtCallbackList	NULL	C
DtNtopCharacter	DtCTextPosition	XmTextPosition	0	CSG
DtNwordWrap	DtCWordWrap	Boolean	True	CSG

継承されるリソース

DtEditor は、次のスーパークラスから動作とリソースを継承します。

- XmForm
- XmManager
- Composite
- Core

詳細は、該当するマニュアル・ページを参照してください。

ローカライズ・リソース

次のリストは、DtEditor ウィジェットとそのダイアログ・ボックスのローカライズのために設計されるウィジェット・リソースのセットを示しています。これらのリソースのデフォルト値は、ロケールに依存します。

- DtNcenterToggleLabel は、[書式の設定] ダイアログ・ボックスの中央に揃えるトグル・ボタンのラベルを指定します。C ロケールでのデフォルト値は、[Center] です。日本語ロケールでは [中央] です。

- DtNchangeAllButtonLabel は、ドキュメントの中の検索文字列のすべての存在箇所を置換する [検索／変更] ダイアログ・ボックスにあるボタンのラベルを指定します。C ロケールでのデフォルト値は、[Change all] です。日本語ロケールでは [すべてを変更] です。
- DtNchangeButtonLabel は、ドキュメントの中の検索文字列の次の存在箇所を置換する [検索／変更] ダイアログ・ボックスにあるボタンのラベルを指定します。C ロケールでのデフォルト値は、[Change] です。日本語ロケールでは [変更] です。
- DtNchangeFieldLabel は、ユーザが置換文字列を指定する [検索／変更] ダイアログ・ボックスにあるフィールドのラベルを指定します。C ロケールでのデフォルト値は、[Change To] です。日本語ロケールでは [変更後の単語] です。
- DtNcurrentLineLabel は、ステータス行の現在の行番号フィールドのラベルを指定します。C ロケールでのデフォルト値は、[line] です。日本語ロケールでは [行] です。
- DtNfindButtonLabel は、ドキュメントの中の検索文字列の次の存在箇所を検索する [検索／変更] ダイアログ・ボックスにあるボタンのラベルを指定します。C ロケールでのデフォルト値は、[Find] です。日本語ロケールでは [検索] です。
- DtNfindChangeDialogTitle は、[検索／変更] ダイアログ・ボックスのタイトルを指定します。DtNdialogTitle がヌルでない場合は、このリソースの前面に追加されてタイトルを形成します。C ロケールでのデフォルト値は、 [Change To] です。日本語ロケールでは [検索／変更] です。
- DtNfindFieldLabel は、ユーザが検索文字列を指定する [検索／変更] ダイアログ・ボックスにあるフィールドのラベルを指定します。C ロケールでのデフォルト値は、[Find] です。日本語ロケールでは [検索] です。
- DtNformatAllButtonLabel は、ドキュメント全体を書式化する [書式の設定] ダイアログ・ボックスにあるボタンのラベルを指定します。C ロケールでのデフォルト値は、[All] です。日本語ロケールでは [すべて] です。
- DtNformatParagraphButtonLabel は、挿入カーソルを含んでいるパラグラフを書式化する [書式の設定] ダイアログ・ボックスにあるボタンのラベルを指定します。C ロケールでのデフォルト値は、[Paragraph] です。日本語ロケールでは [パラグラフ] です。
- DtNformatSettingsDialogTitle は、[書式の設定] ダイアログ・ボックスのタイトルを指定します。DtNdialogTitle がヌルでない場合は、このリソースの前面に追加されてタイトルを形成します。C ロケールでのデフォルト値は、[Format Setting] です。日本語ロケールでは [書式の設定] です。

- DtNinformationDialogTitle は、ユーザにフィードバックと一般情報を提示するのに使用される [インフォメーション] ダイアログ・ボックスのタイトルを指定します。DtNdialogTitle がヌルでない場合は、このリソースの前面に追加されてタイトルを形成します。C ロケールでのデフォルト値は、[Infomation] です。日本語ロケールでは [インフォメーション] です。
- DtNjustifyToggleLabel は、[書式の設定] ダイアログ・ボックスにある両端揃えトグル・ボタンのラベルを指定します。C ロケールでのデフォルト値は、[Justify] です。日本語ロケールでは [両端揃え] です。
- DtNleftAlignToggleLabel は、[書式の設定] ダイアログ・ボックスにある左揃えトグル・ボタンのラベルを指定します。C ロケールでのデフォルト値は、[Left Align] です。日本語ロケールでは [左揃え] です。
- DtNleftMarginFieldLabel は、[書式の設定] ダイアログ・ボックスにある左マージン値フィールドのラベルを指定します。C ロケールでのデフォルト値は、[Left Margin] です。日本語ロケールでは [左マージン] です。
- DtNmisspelledListLabel は、[スペルチェック] ダイアログ・ボックスにある認識できない、またはスペルが間違っている単語のリストのラベルを指定します。C ロケールでのデフォルト値は、[Misspelled Words] です。日本語ロケールでは [スペルミスの単語] です。
- DtNoverstrikeLabel は、エディタが上書きモードであることを示すステータス行にあるラベルを指定します。C ロケールでのデフォルト値は、[Overstrike] です。日本語ロケールでは [上書き] です。
- DtNrightAlignToggleLabel は、[書式の設定] ダイアログ・ボックスにある右揃えトグル・ボタンのラベルを指定します。C ロケールでのデフォルト値は、[Right Align] です。日本語ロケールでは [右揃え] です。
- DtNrightMarginFieldLabel は、[書式の設定] ダイアログ・ボックスにある右マージン値フィールドのラベルを指定します。C ロケールでのデフォルト値は、[Right Margin] です。日本語ロケールでは [右マージン] です。
- DtNspellDialogTitle は、[書式の設定] ダイアログ・ボックスのタイトルを指定します。DtNdialogTitle がヌルでない場合は、このリソースの前方に追加されてタイトルを形成します。C ロケールでのデフォルト値は、[Spell] です。日本語ロケールでは [スペルチェック] です。
- DtNtotalLineCountLabel は、ドキュメントの総行数を示すステータス行にあるディスプレイのラベルを指定します。C ロケールでのデフォルト値は、[Total] です。日本語ロケールでは [合計] です。

ローカライズ・リソースのそれぞれのクラス、型、デフォルト、およびアクセスを表 7–10 にリストします。アクセス欄のコードは、次の作業が可能かどうかを示します。

- 作成時にリソースを設定する (C)
- `XtSetvalues` を使用して設定する (S)
- `XtGetValues` を使用して検索する (G)

詳細は、`DtEditor(3)` のマニュアル・ページを参照してください。

表 7–10 DtEditor ローカライズ・リソース

名前	クラス	型	デフォルト	アクセス
`DtNcenterToggleLabel`	`DtCCenterToggleLabel`	XmString	Dynamic	CSG
`DtNchangeAllButtonLabel`	`DtCChangeAllButtonLabel`	XmString	Dynamic	CSG
`DtNchangeButtonLabel`	`DtCChangeButtonLabel`	XmString	Dynamic	CSG
`DtNchangeFieldLabel`	`DtCChangeFieldLabel`	XmString	Dynamic	CSG
`DtNcurrentLineLabel`	`DtCCurrentLineLabel`	XmString	Dynamic	CSG
`DtNfindButtonLabel`	`DtCFindButtonLabel`	XmString	Dynamic	CSG
`DtNfindChangeDialogTitle`	`DtCFindChangeDialogTitle`	XmString	Dynamic	CSG
`DtNfindFieldLabel`	`DtCFindFieldLabel`	XmString	Dynamic	CSG
`DtNformatAllButtonLabel`	`DtCFormatAllButtonLabel`	XmString	Dynamic	CSG
`DtNformatParagraphButtonLabel`	`DtCFormatParagraphButtonLabel`	XmString	Dynamic	CSG
`DtNformatSettingsDialogTitle`	`DtCFormatSettingsDialogTitle`	XmString	Dynamic	CSG
`DtNinformationDialogTitle`	`DtCInformationDialogTitle`	XmString	Dynamic	CSG
`DtNjustifyToggleLabel`	`DtCJustifyToggleLabel`	XmString	Dynamic	CSG

表 7–10　DtEditor ローカライズ・リソース　続く

名前	クラス	型	デフォルト	アクセス
DtNleftAlignToggleLabel	DtCLeftAlignToggleLabel	XmString	Dynamic	CSG
DtNleftMarginFieldLabel	DtCLeftMarginFieldLabel	XmString	Dynamic	CSG
DtNmisspelledListLabel	DtCMisspelledListLabel	XmString	Dynamic	CSG
DtNoverstrikeLabel	DtCOverstrikeLabel	XmString	Dynamic	CSG
DtNrightAlignToggleLabel	DtCRightAlignToggleLabel	XmString	Dynamic	CSG
DtNrightMarginFieldLabel	DtCRightMarginFieldLabel	XmString	Dynamic	CSG
DtNspellDialogTitle	DtCSpellDialogTitle	XmString	Dynamic	CSG
DtNtotalLineCountLabel	DtCTotalLineCountLabel	XmString	Dynamic	CSG

コールバック関数

DtEditor ウィジェットは、次の 3 つのコールバック関数をサポートします。

- DtEditorNHelpCallback
- DtNtextSelectCallback
- DtNtextDeselectCallback

エディタ・ウィジェットとそのダイアログ・ボックスについてのヘルプ情報を表示する場合は、XmNhelpCallback リソースを設定
し、DtEditorHelpCallbackStruct の一部として渡される reason フィールドを使用して、[ヘルプ] ダイアログ・ボックスの内容を設定します。次の構造体へのポインタが XmNHelpCallback に渡されます。コールバックのための構造体を次に示し、表 7–11 で説明します。

```
typedef struct {
  int      reason;
  XEvent   *event;
} DtEditorHelpCallbackStruct;
```

表 7–11 DtEditorHelp コールバックのための構造体

構造体	説明
reason	コールバックが呼び出された reason。reason のリストについては、DtEditor(3) のマニュアル・ページを参照してください。
event	このコールバックを呼び出したイベントへのポインタ。値は、NULL になることもあります。

テキストが選択されているかどうかによって、メニュー項目とコマンドを有効か無効にする場合は、DtNtextSelectCallback リソースおよび DtNtextDeselectCallback リソースを使用します。DtNtextSelectCallback は、編集ウィンドウでテキストが選択されたときに呼び出される関数を指定します。DtNtextDeselectCallback は、編集ウィンドウでテキストが選択されていないときに呼び出される関数を指定します。コールバックによって送られる reason は、DtEDITOR_TEXT_SELECT と DtEDITOR_TEXT_DESELECT です。

第 8 章

アプリケーションからのアクションの実行

アプリケーションが拡張性のある一連のデータ型を管理する場合には、アクションの実行によりデータ型を直接実行しなければなりません。この章では、アプリケーションからアクションを実行する方法について説明します。アクションの実行方法を示すサンプル・プログラムも示します。

アクションとアクションの作成の詳細は、第 9 章「データ型データベースのアクセス」と、『*Solaris* 共通デスクトップ環境 上級ユーザ及びシステム管理者ガイド』の次の章を参照してください。

- 『*Solaris* 共通デスクトップ環境 上級ユーザ及びシステム管理者ガイド』の「アクションとデータ型の概要」
- 『*Solaris* 共通デスクトップ環境 上級ユーザ及びシステム管理者ガイド』の「アクション作成ツールを使ったアクションとデータ型の作成」
- 『*Solaris* 共通デスクトップ環境 上級ユーザ及びシステム管理者ガイド』の「手入力によるアクションの作成」
- 『*Solaris* 共通デスクトップ環境 上級ユーザ及びシステム管理者ガイド』の「手入力によるデータ型の作成」
- 114ページの「アプリケーションからアクションを実行する方法」
- 115ページの「アクションの型」
- 116ページの「アクション実行 API」
- 116ページの「関連情報」
- 117ページの「actions.c プログラム例」
- 124ページの「actions.c のリスト」

アプリケーションからアクションを実行する方法

デスクトップ・サービス・ライブラリによってエクスポートされたアクション実行 API は、アプリケーションから別のアプリケーションを実行したり、操作を実行したりするための方法の 1 つです。その他の方法として、次のものがあります。

- `fork/exec` システム・コール
- ToolTalk メッセージ

これらの方法は、それぞれ利点と制約があるので、具体的な状況を評価して、どちらが適切かを判断しなければなりません。

アクション実行 API の利点は、次のとおりです。

- アクションは、従来のコマンド行アプリケーション (すなわち、`COMMAND` アクション) と ToolTalk アプリケーション (すなわち、`TT_MSG` アクション) の両方をカプセル化できます。アクションを実行するアプリケーションは、コマンドがフォークされたのか、それともメッセージが送られたのかを知る必要はありません。
- アクションは多様性を持ち、デスクトップのデータ型機構と統合されます。これは、[開く] や [印刷] などのアクションは、与えられる引き数の型に基づいて異なる動作をするが、動作の違いは、アクションを呼び出すアプリケーションに対して透過されることです。
- アクションは、アプリケーション開発者、システム統合者、システム管理者、およびエンドユーザに対して、構成の大きな可能性を提供します。これらのユーザは、アクション・データベースを編集して、アクションの実行方法の定義を変更できます。
- アクションは、分散環境でも有効です。アプリケーションが `fork/exec` により別のアプリケーションを直接実行する場合には、両方のアプリケーションが同じシステム上で使用可能でなければならず、同じシステム上で実行可能でなければなりません。それに対して、アクション実行 API は、アクション・データベース内の情報に基づいて、どのシステム上で `COMMAND` アクションを実行するかを判断します。
- アクションによって、デスクトップの動作と常に一貫性のあるアプリケーションの動作が可能になります。これは、デスクトップのコンポーネントがユーザのデータ・ファイルを操作するときに、アクションを使用することで対話するからです。

アクション実行 API の欠点は、戻り値機能が制限されている実行方法であり、実行されたアクション・ハンドラとの対話機能がないことです。これらの機能が必要な場合には、`fork/exec/pipes` を使用できます。ただし、共通デスクトップ環境 (CDE) で望ましいプロセス間通信の方法は、一般化されたクライアント／サーバ・パラダイムを持つ ToolTalk です。

実行について説明します。アプリケーションがいくつかの異なる形式(テキストとグラフィック) のデータ・ファイルを管理すると仮定し、これらのファイルの編集と表示の手段をユーザに提供する必要があると仮定します。アクションを使用せずにこれを実現するには、次の方法の 1 つを使用することになります。

- `fork/exec` を使用して、適切なエディタを起動し、ユーザがエディタの名前を指定するための何らかの方法 (環境変数など) を考えてください。このアプローチには次のような制約があります。
  - システム・コールによりサブプロセスを実行し、その結果のシグナルを監視する複雑なコードを書かなければなりません。
  - アプリケーションと同じシステム上で使用できるエディタが必要であり、システム管理者は、`rsh` などの機能を使用する複雑な構成を提供しなければなりません。
  - システム管理者とユーザは、アプリケーションの固有の構成モデルを学び、管理しなければなりません。
- ToolTalk メッセージを使用して、編集や表示などの操作をデータに対して実行することを要求します。このアプローチには、すべてのデータ型に対して使用可能な ToolTalk 形式のエディタが必要であるという制約があります。

アクションによりこれを実現するには、バッファまたはデータ・ファイルに対して [開く] アクションを実行するだけです。アクション実行 API はアクション・データベースに基づいて、送信する適切なメッセージまたは実行するコマンドを判断し、一時ファイルの作成や削除、必要なシグナルの取り込みなどのすべての詳細を処理します。

アクションの型

アクションのアプリケーション・プログラム・インタフェース (API) は、どの種類のアクションに対しても機能します。デスクトップでのアクションの種類は、次のとおりです。

コマンド・アクション	実行するコマンド行を指定します。
ToolTalk アクション	送信する ToolTalk メッセージを指定します。メッセージは、適切なアプリケーションによって受信されます。
マップ・アクション	特定の動作を定義する代わりに、別のアクションを参照します。

詳細は、『*Solaris* 共通デスクトップ環境 上級ユーザ及びシステム管理者ガイド』の第 10 章「アクションとデータ型の概要」を参照してください。

アクション実行 API

アクション実行 API は、デスクトップ・サービス・ライブラリからエクスポートされて、次のような多数のタスクを実行する関数を提供します。

- アクションおよびデータ型定義のデータベースを初期化し、読み込みます。アクションを実行するためには、その前にデータベースが読み込まれていなければなりません。
- データベースに問い合わせます。指定されたアクション、アクションに関連付けられたアイコン・イメージ、ラベル、または記述が存在するかどうかを判断する関数があります。
- アクションを実行します。アプリケーションは、ファイルまたはバッファ引き数をアクションに渡すことができます。
- アクション・ステータスを受け取り、引き数を返すコールバックを登録します。

関連情報

アクション・コマンド、関数、およびデータ形式の詳細は、次のマニュアル・ページを参照してください。

- `dtaction(1)`
- `dtactionfile(4)`
- `DtActionCallbackProc(3)`

- DtActionDescription(3)
- DtActionExists(3)
- DtActionIcon(3)
- DtActionInvoke(3)
- DtActionLabel(3)
- DtActionQuit(3)
- DtActionQuitType(3)
- DtActionStUpCb(3)
- dtexec(1)

actions.c プログラム例

この節では、簡単なサンプル・プログラム actions.c について説明します。actions.c の完全なリストは、この章の終わりにあります。

アクションおよびデータ型データベースの読み込み

アプリケーションがアクションを実行するには、その前に、デスクトップ・サービス・ライブラリ (アクション実行 API を含む) を初期化して、アクションおよびデータ型定義のデータベースを読み込まなければなりません。

▼ デスクトップ・サービス・ライブラリを初期化するには

♦ デスクトップ・サービス・ライブラリを初期化するには、DtInitialize() 関数を使用します。

```
DtInitialize(*display,widget,*name,*tool_class)
```

DtInitialize() は、デフォルトのイントリンシクス関数 XtAppContext を使用します。API は、アプリケーションが *app_context* を指定しなければならないときに使用する追加の関数 DtAppInitialize() を提供します。

```
DtAppInitialize(app_context,*display,widget,*name,  tool_class)
```

DtInitialize() の例

次のコードの一部分は、サンプル・プログラム actions.c の中で DtInitialize() がどのように使用されているかを示しています。

```
    if (DtInitialize(XtDisplay(shell), shell,

argv[0],ApplicationClass)==False) {
    /* DtInitialize() has already logged an appropriate error msg */
            exit(-1);
    }
```

▼ アクションおよびデータ型データベースを読み込むには

- アクションおよびデータ型データベースを読み込むには、DtDbLoad() 関数を使用します。

```
DtDbLoad(void)
```

DtDbLoad() は、アクションおよびデータ型データベースを読み込みます。この関数は、データベース・ファイルを検索するディレクトリのセット (データベース検索パス) を判断して、データベース内で見つかった *.dt ファイルを読み込みます。ディレクトリ検索パスは、DTDATABASESEARCHPATH 環境変数と内部のデフォルト値に基づきます。

▼ 再読み込みイベントの通知を要求するには

長時間実行中のアプリケーションの中で DtDbLoad() を使用する場合、データベースが変更されたときには、動的に再読み込みしなければなりません。

1. DtDbReloadNotify() 関数を使用して、再読み込みイベントの通知を要求します。

```
/* Notice changes to the database without needing to restart
  application */

   DtDbReloadNotify(DbReloadCallbackProc, callback_proc,
   XTPointer, client_data);
```

2. 次の作業を実行するコールバックを指定します。

- アプリケーションによって保持されている、キャッシュされたデータベース情報を破棄する。
- DtDbLoad() 関数を再コールする。

callback_proc は、アプリケーションが保持している、キャッシュされたデータベース情報をクリーンアップしてから、DtDbLoad() を呼び出します。*client_data* を使用して、追加のクライアント情報をコールバック・ルーチンに渡すことができます。

アクション・データベースのチェック

アプリケーションは、アクションのアイコンまたはラベルを表示する必要がある場合には、データベースにアクセスします。また、アクションを実行することによって、アプリケーションはアクションの存在をチェックできます。データベース内のアクションは、アクション名によって識別されます。

```
ACTION action_name
{
  …
}
```

たとえば、[電卓] アクションの定義は次のとおりです。

```
ACTION Dtcalc
{
      LABEL     電卓
      ICON       Dtcalc
      ARG_COUNT       0
      TYPE             COMMAND
      WINDOW_TYPE       NO_STDIO
      EXEC_STRING       /usr/dt/bin/dtcalc
      DESCRIPTION       電卓 (Dtcalc) アクションは、デスクトップ電卓 \
                          アプリケーションを起動します
}
```

[電卓] アクションのアクション名は Dtcalc です。

実行形式ファイルがデータベース内のアクション名と一致するファイル名を持つ場合には、そのファイルはアクション・ファイルです。すなわち、基本のアクションの表現です。そのファイルのアイコンとラベルに関する情報は、データベースに格納されます。

▼ 指定されたアクション定義が存在するかどうかを判断するには

- 指定されたアクション定義が存在するかどうかを判断するには、DtActionExists() 関数を使用します。

```
DtActionExists(*name)
```

DtActionExists() は、指定された名前がデータベース内のアクションの名前に一致するかどうかをチェックします。この関数は、名前がアクション名に一致する場合には True を返し、その名前のアクションが見つからない場合には False を返します。

▼ 指定されたアクションのアイコン・イメージ情報を取り出すには

- アイコン・イメージ情報を取り出すには、DtActionIcon() 関数を使用します。

```
DtActionIcon(char *action_name)
```

アクション定義は、アクションを表すために使われるアイコン・イメージを定義の ICON フィールドで指定します。

```
ACTION action_name
{
      ICON icon_image_base_name
      …
}
```

DtActionIcon() は、アイコン・イメージ・フィールドの値にある文字列を返します。アクション定義にアイコン・フィールドがない場合には、この関数はデフォルトのアクション・アイコン・イメージの値 Dtactn を返します。

次に、使用したいアイコンとサイズの位置を決めます。アイコンには 4 つのサイズがあり、ビットマップまたはピックスマップ形式で使用できます。たとえば、[電卓] のアクション定義からアイコン・ファイルのベース名を見つけることができます。次に、そのベース名と表 8–1 の情報の組み合わせと、すべてのアイコンの格納情報から、目的のアイコン・ファイルを見つけ出せます。

[電卓] アクションのアイコン名は Dtcalc ですが、これはファイル名全体ではありません。アイコン・ファイル名はアイコンのサイズに基づき、4 つのサイズがあり

ます。表 8–1 は、デスクトップ・アイコンのサイズとファイル名の命名規則を示します。

表 8–1 アイコンのサイズとファイル名

アイコンのサイズ	ビットマップ名	ピックスマップ名
16 × 16 (極小)	*name*.t.bm	*name*.t.pm
24 × 24 (小)	*name*.s.bm	*name*.s.pm
32 × 32 (中)	*name*.m.bm	*name*.m.pm
48 × 48 (大)	*name*.l.bm	*name*.l.pm

注 - デスクトップ・アイコン・ファイルの詳細は、『*Solaris* 共通デスクトップ環境上級ユーザ及びシステム管理者ガイド』の第 14 章「デスクトップのアイコンの作成」を参照してください。

ビットマップの場合、マスクとして使われる追加のファイルがあり、そのファイルの拡張子 _m.bm で終わります。したがって、各サイズのアイコンに対して合計 3 個のファイルがあります。次に、電卓のアイコン・ファイルを示します。

Dtcalc.t.bm
Dtcalc.t.pm
Dtcalc.t_m.bm
Dtcalc.m.bm
Dtcalc.m.pm
Dtcalc.m_m.bm
Dtcalc.l.bm
Dtcalc.l.pm
Dtcalc.l_m.bm

注 - 電卓には小型アイコン (Dtcalc.s.bm、Dtcalc.s.pm、Dtcalc.s_m.bm) がない点に注意してください。

DtActionIcon() はベース名だけを返します。電卓の場合は Dtcalc です。種類 (ピックスマップまたはビットマップ) とサイズ (極小、小、中、大) を選択して、適用可能な拡張子をベース名に追加してください。また、ファイルがどこにあるかを知っておいてください。

▼ アクションのローカライズ・ラベルを取り出すには

- **アクションのローカライズ・ラベルを取り出すには、`DtActionLabel()` 関数を使用します。**

```
char *DtActionLabel(char *actionName)
```

アクション定義にはラベルを入れることができます。ラベルは、*label_text* フィールドを使用して定義されます。

```
ACTION action_name {
   LABEL label_text
   …
}
```

このラベルは、グラフィック・コンポーネント (ファイル・マネージャやアプリケーション・マネージャなど) の中でアクションのアイコンにラベルを付けるために使用されます。アクション定義に *label_text* フィールドがない場合には、*action_name* が使用されます。

label_text 文字列の値は、エンドユーザがアクションを見分けられるように、すべてのインタフェース・コンポーネントによって使用されなければなりません。

`DtActionLabel()` 関数は、*actionName* という名前のアクションのアクション定義の中の *label_text* フィールドの値を返します。*label_text* フィールドがない場合には、この関数は *actionName* を返します。

アクションの実行

アプリケーションがデスクトップ・サービス・ライブラリを初期化した後は、アクションを実行できます。

▼ アクションを実行するには

- **アクションを実行するには、`DtActionInvoke()` 関数を使用します。**

`DtActionInvoke` (*widget*, *action*, *args*, *argCount*, *termOpts*, *execHost*,, *contexDir*, *useIndicator*,*statusUpdateCb*, *client_data*)

`DtActionInvoke()` は、アクション・データベースから、指定されたアクション名に一致するエントリを探して、指定されたクラス、型、およびカウントの引き数を受け入れます。アクションを実行する前に、アプリケーションはデータベースを初期化し、読み込まなければならないので注意してください。

actions.c のリスト

```
/*
 * (c) Copyright 1993, 1994 Hewlett-Packard Company
 * (c) Copyright 1993, 1994 International Business Machines Corp.
 * (c) Copyright 1993, 1994 Sun Microsystems, Inc.
 * (c) Copyright 1993, 1994 Novell, Inc.
 */

#include <Xm/XmAll.h>
#include <Dt/Dt.h>
#include <Dt/Action.h>

#define ApplicationClass "Dtaction"

static Widget shell;
static XtAppContext appContext;
static Widget actionText;
static Widget fileText;

static void CreateWidgets(Widget);
static void InvokeActionCb(Widget, XtPointer, XtPointer);
static void InvokeAction(char*, char*);
static void DbReloadProc(XtPointer);


void main(int argc, char **argv)
{
   Arg args[20];
   int n=0;
   int numArgs = 0;

    shell = XtAppInitialize(&appContext , ApplicationClass, NULL, 0,
      &argc, argv, NULL, args, n);

    CreateWidgets(shell);

   if (DtInitialize(XtDisplay(shell), shell, argv[0],
  ApplicationClass)==False) {
      /* DtInitialize() has already logged an appropriate error msg */
      exit(-1);
   }
```

```
    /* Load the filetype/action databases */
    DtDbLoad();
    /* Notice changes to the database without needing to restart application
 */
    DtDbReloadNotify(DbReloadProc, NULL);
    XtRealizeWidget(shell);
    XmProcessTraversal(actionText, XmTRAVERSE_CURRENT);

    XtAppMainLoop(appContext);
}

static void CreateWidgets(Widget shell)
{
    Widget messageBox, workArea, w;
    Arg args[20];
    int n;
    XmString labelString;
     labelString = XmStringCreateLocalized("Invoke");
     n = 0;
    XtSetArg(args[n], XmNdialogType, XmDIALOG_TEMPLATE); n++;
    XtSetArg(args[n], XmNokLabelString, labelString); n++;
    messageBox = XmCreateMessageBox(shell, "messageBox", args, n);
    XtManageChild(messageBox);
    XmStringFree(labelString);
    XtAddCallback(messageBox, XmNokCallback, InvokeActionCb, NULL);

    n = 0;
    XtSetArg(args[n], XmNorientation, XmVERTICAL); n++;
    XtSetArg(args[n], XmNpacking, XmPACK_COLUMN); n++;
    XtSetArg(args[n], XmNnumColumns, 2); n++;
    XtSetArg(args[n], XmNentryAlignment, XmALIGNMENT_END); n++;
    workArea = XmCreateWorkArea(messageBox, "workArea", args, n);
    XtManageChild(workArea);

    labelString = XmStringCreateLocalized("Invoke Action:");
    n = 0;
    XtSetArg(args[n], XmNlabelString, labelString); n++;
    w = XmCreateLabel(workArea, "actionLabel", args, n);
    XtManageChild(w);
    XmStringFree(labelString);

    labelString = XmStringCreateLocalized("On File:");
    n = 0;    XtSetArg(args[n], XmNlabelString, labelString); n++;
    w = XmCreateLabel(workArea, "fileLabel", args, n);
    XtManageChild(w);
    XmStringFree(labelString);

    n = 0;
    XtSetArg(args[n], XmNcolumns, 12); n++;
    actionText = XmCreateTextField(workArea, "actionText", args, n);
    XtManageChild(actionText);

    n = 0;
    XtSetArg(args[n], XmNcolumns, 12); n++;
    fileText = XmCreateTextField(workArea, "fileText", args, n);
    XtManageChild(fileText);
}
```

```
static void DbReloadProc(XtPointer cd)
{
   /* Pick up any dynamic changes to the database files */
   DtDbLoad();
}

static void InvokeActionCb(Widget w, XtPointer cd, XtPointer cb)
{
   char *action;
   char *file;

   action = XmTextFieldGetString(actionText);
   if (action == NULL) return;
   if (strlen(action) == 0) {
      XtFree(action);
      return;
   }

   file = XmTextFieldGetString(fileText);

   InvokeAction(action, file);

   XtFree(action);
   XtFree(file);

   XmTextFieldSetString(actionText, "");
   XmTextFieldSetString(fileText, "");
   XmProcessTraversal(actionText, XmTRAVERSE_CURRENT);
}

static void InvokeAction(char *action, char *file)
{
   DtActionArg *ap = NULL;

int nap = 0;
   DtActionInvocationID actionId;

    /* If a file was specified, build the file argument list */

printf("%s(%s)\n",action,file);
   if (file != NULL && strlen(file) != 0) {
      ap = (DtActionArg*) XtCalloc(1, sizeof(DtActionArg));
      ap[0].argClass = DtACTION_FILE;
      ap[0].u.file.name = file;
      nap = 1;
   }

   /* Invoke the specified action */

  actionId = DtActionInvoke(shell,action,ap,nap,NULL,NULL,NULL,True,NULL,NULL);

}
```

第 9 章

データ型データベースのアクセス

この章では、データ型関数とデータ型データベースの使い方について説明します。

要約

データ型により、従来の UNIX ファイル・システムによって提供される機能を越えて、ファイルとデータの属性が拡張されます。これらの拡張は、アイコン名、記述、アクションなどの属性から成っており、ファイルがデータ上で実行できます。この情報は、DATA_ATTRIBUTES テーブル (またはデータベース) に名前と値の対として格納されます。デスクトップは、次のパラグラフで説明する特定の DATA_ATTRIBUTES のセットを使用します。DATA_ATTRIBUTES テーブルは、将来およびアプリケーション固有の成長のために拡張可能ですが、他のアプリケーションでは追加をチェックできないので、このテーブルを拡張することは推奨しません。

データを、特定のファイルまたは DATA_CRITERIA テーブルのデータ・エントリに一致させます。DATA_CRITERIA テーブルのエントリは、具体性が高いものから具体性が低いものへ降順で並べられます。たとえば、/usr/lib/lib* は /usr/* よりも具体的なので、/usr/* より前に置かれます。ファイルまたはデータの型の検

査が要求されると、このテーブルが始めから順にチェックされ、ファイルまたはデータから与えられた情報を使用して最も一致するものが検索されます。情報に一致するエントリが見つかると、DATA_ATTRIBUTES_NAME を使用して、正しい DATA_ATTRIBUTES エントリが検索されます。

アプリケーションがデスクトップと同じ方法でデータ・オブジェクト (ファイルまたはデータ・バッファ) をユーザに提示するようにする場合は、DtDts* API を使用して、データ・オブジェクトの表示方法とデータ・オブジェクトの操作方法を指定します。たとえば、アプリケーションは、ICON 属性に対して DtDtsDataTypeToAttributeValue() 関数を呼び出すことによって、データ・オブジェクトを表すアイコンを判断できます。

ライブラリとヘッダ・ファイル

データ型を使用するには、libDtSvc ライブラリをリンクしてください。アクションは、通常はデータ型情報と一緒に読み込まれます。アクションは、libXm ライブラリと libX11 ライブラリのリンクを必要とします。ヘッダ・ファイルは、Dt/Dts.h と Dt/Dt.h です。

デモ・プログラム

データ型データベースの使用例が入っているデモ・プログラムが、/usr/dt/examples/dtdts/datatypes/datatyping.c にあります。

データの基準とデータの属性

データ型検査は、次の 2 つの部分から成ります。

- データの基準とデータの属性を格納するデータベース
- データベースに問い合わせるルーチンの集まり

データ基準の属性は、次のとおりです (アルファベット順)。

- CONTENT
- DATA_ATTRIBUTES_NAME
- LINK_NAME

- LINK_PATH
- MODE
- NAME_PATTERN
- PATH_PATTERN

データの基準を使用頻度が高いものから順に表 9–1 に示します。

表 9–1 データの基準 (使用頻度順)

基準	説明	使用例
DATA_ATTRIBUTES_NAME	このデータ型の名前。この値は、データ属性テーブルの中の record_name です。	POSTSCRIPT
NAME_PATTERN	このデータに一致するファイル名を記述するシェル・パターン照合表現。デフォルトは空の文字列で、照合の際にファイル名のパターンを無視することを意味します。	*.ps
CONTENT	ファイル・ユーティリティが使用し、マジック・ファイルの開始、型、および値のフィールドとして解釈される 3 つの値。詳細は、file(1) のマニュアル・ページを参照してください。デフォルトは空のフィールドで、照合の際に内容を無視することを意味します。一致する型の例としては、文字列、バイト、ショート、ロング、およびファイル名があります。	0 string !%
MODE	stat 構造体のモード・フィールドに一致する 0 ～ 4 文字の文字列。詳細は、stat(2) のマニュアル・ページを参照してください。最初の文字は、次のとおりです。 d は、ディレクトリに一致します。 s は、ソケットに一致します。 l は、シンボリック・リンクに一致します。 f は、通常ファイルに一致します。 b は、ブロック・ファイルに一致します。 c は、文字型特殊ファイルに一致します。	f&!x

表 9–1 データの基準 (使用頻度順) 続く

基準	説明	使用例
	次の文字は、最初または後続の文字にできます。 r は、ユーザ、グループ、またはその他の読み取り権ビットが設定されたファイルに一致します。 w は、ユーザ、グループ、またはその他の書き込み権ビットが設定されたファイルに一致します。 x は、ユーザ、グループ、またはその他の実行あるいはディレクトリ検索のアクセス権ビットが設定されているファイルに一致します。 たとえば、frw の MODE フィールドは、読み取り可能または書き込み可能な通常ファイルに一致します。x は、実行可能なビットまたは検索ビットが設定されたファイルに一致します。 デフォルトは空のフィールドで、照合の際にモードを無視することを意味します。	
PATH_PATTERN	このデータに一致する絶対パス名を記述するシェル・パターン照合式。デフォルトは空の文字列で、照合の際にパス・パターンを無視することを意味します。	*/*mysubdir*/*
LINK_NAME	dtdtsfile(4) のマニュアル・ページを参照してください。	
LINK_PATH	dtdtsfile(4) のマニュアル・ページを参照してください。	

データ型の一般的な属性のいくつかをアルファベット順に示します。

- ACTIONS
- COPY_TO_ACTION
- DESCRIPTION
- ICON
- INSTANCE_ICON
- IS_EXECUTABLE
- IS_TEXT

- LINK_TO_ACTION
- MEDIA
- MIME_TYPE
- MOVE_TO_ACTION
- NAME_TEMPLATE
- PROPERTIES
- X400_TYPE

これらのデータの属性を使用頻度が高い順に表 9–2 に示します。

表 9–2 データの属性 (使用頻度順)

基準	説明	使用例
DESCRIPTION	人間が読める形式で書かれたデータの説明。このフィールドが NULL か、データ属性レコードに含まれていない場合は、データ属性名が使用されます。	This is a PostScript page description.
ICON	このデータに対して使用されるアイコン名。このフィールドが NULL か、データ属性レコードに含まれていない場合は、標準のアイコンが使用されます。アイコンの命名の詳細は、dtdtsfile(4) のマニュアルページを参照してください。	Dtps
PROPERTIES	このデータの属性を示すキーワード。有効な値は、見える場合と見えない場合があります。このフィールドが NULL か、データ属性レコードに含まれていない場合は、可視属性とみなされます。これは、ファイルをユーザから完全に隠したい場合に使用します。	invisible
ACTIONS	このデータに対して実行できるアクションのリスト。このリストは、この型のオブジェクトに対してユーザに提示されるアクションのアクション・テーブル内の名前を参照します。このフィールドが NULL か、データ属性レコードに含まれていない場合は、どのアクションも使用できません。	Open,Print

表 9–2 データの属性 (使用頻度順) 続く

基準	説明	使用例
NAME_TEMPLATE フィールド	この型のデータの新規ファイル作成に使用される文字列。文字列は、ファイル名と共に 1 つの引き数として sprintf(3) に渡されます。デフォルトは空です。このフィールドをデータ抽出条件テーブルの NAME_PATTERN フィールドと比較してみてください。テンプレートは %s.c など、特定のファイルを作成するために使用されますが、パターンは *.c などのファイルを検索するために使用されます。	%s.ps
IS_EXECUTABLE フィールド	このデータ型をアプリケーションとして実行できることをユーザに知らせる文字列論理値。IS_EXECUTABLE に true が設定されている場合 (DtDtsIsTrue() 参照)、データは実行可能です。このフィールドが NULL かデータ属性レコードに含まれていない、または true に設定されていない場合は、データは実行可能ではないとみなされます。	true
MOVE_TO_ACTION	オブジェクトが現在のオブジェクトに移動されるときに実行されるアクション名	FILESYSTEM_MOVE
COPY_TO_ACTION	オブジェクトが現在のオブジェクトにコピーされるときに実行されるアクション名	FILESYSTEM_COPY
LINK_TO_ACTION	オブジェクトが現在のオブジェクトにリンクされるときに実行されるアクション名	FILESYSTEM_LINK
IS_TEXT	このデータ型がテキスト・エディタまたはテキスト・ウィジェットでの操作 (表示または編集) に適していることをユーザに知らせる文字列論理値。データが本来はテキストである場合や、ユーザに対してテキスト形式で表示される場合、IS_TEXT フィールドには true が設定されます (DtDtsIsTrue() 参照)。その基準は、データが人間の言語から成るものか、手動で生成および管理されているか、テキスト・エディタでの表示と編集が可能か、構造体と書式の情報をまったく (あるいはごくわずかしか) ないかどうかという点から決定されます。	詳細な例については、表 9–3 を参照してください。

表 9–2 データの属性 (使用頻度順) 続く

基準	説明	使用例
	IS_TEXT フィールドが true の場合、データはアプリケーションから直接表示できます。すなわち、アプリケーションは XmText などのテキスト編集ウィジェットにデータを直接読み込むことができます。	
MEDIA フィールド	MEDIA ネーム・スペース名は、データそのものの形式について記述します。MEDIA 名は、ICCCM 選択ターゲットとして使用され、データ型レコードの MEDIA フィールドで名前が付けられ、ToolTalk メディア交換メッセージの型パラメータの中で使用されます。 MEDIA ネーム・スペースは、ICCCM によって定義された選択ターゲット・アトムのネーム・スペースのサブセットです。データ書式を指定する選択ターゲットは、すべて有効な MEDIA 名です。有効な MEDIA 名は選択ターゲットとして直接使用できます。データ書式ではなく、選択の属性 (たとえば、LIST_LENGTH) や発生する副作用 (たとえば、DELETE) を指定する選択ターゲットもあります。これらの属性選択ターゲットは、MEDIA ネーム・スペースの一部ではありません。	POSTSCRIPT
MIME_TYPE	MEDIA は、デスクトップ内部にあり、データ型を表す一意の名前です。ただし、外部の他の命名組織もネーム・スペースを設定しています。MIME RFC で述べられている Multipurpose Internet Message Extensions (MIME)は、そのような外部登録の 1 つであり、デスクトップ・メール・プログラムのための標準的なネーム・スペースです。	application/postscript
X400_TYPE	X.400 型は、構造は MEDIA 型に似ていますが、異なる規則を使用して書式化され、異なる命名組織を持ちます。	1 2 840 113556 3 2 850

表 9–2 データの属性 (使用頻度順) 続く

基準	説明	使用例
INSTANCE_ICON フィールド	データのインスタンスのために使用されるアイコン名で、通常は %*name*%.icon などの値 (dtdtsfile(4) のマニュアル・ページの「バグ」も参照)。INSTANCE_ICON が設定されている場合は、アプリケーションは ICON の代わりに使用しなければなりません。このフィールドが NULL か、データ属性レコードに含まれていない場合は、ICON フィールドが使用されます。	*/myicondir*/%*name*%.bm
DATA_HOST	DATA_HOST 属性は、*.dt ファイルのデータ属性テーブルに追加できるフィールドではありませんが、テーブルから属性を読み込むアプリケーションに返すことができます。データ型検査サービスはこの属性を自動的に追加して、データ型の読み込み元のホスト・システムを示します。このフィールドが NULL か、データ属性レコードに含まれていない場合、データ型はローカル・システムから読み込まれています。	

IS_TEXT フィールドは、MIME_RFC で述べられている MIME コンテント・タイプである MIME_TYPE フィールドのテキスト属性とは異なります。MIME コンテント・タイプから、データがテキスト文字とバイト値のどちらで作成されているかがわかります。データがテキスト文字で作成され、データに text/* というラベルが付けられている場合、IS_TEXT フィールドはテキスト形式でユーザに表示するのに適したデータかどうかを判別します。

さまざまな MIME_TYPE属性での IS_TEXTの使用例を表 9–3 に示します。

表 9–3 IS_TEXT 属性の例

説明と **MIME_TYPE** 属性	**IS_TEXT** 値
ASCII でコード化された人間の言語 (MIME_TYPE text/plain)	IS_TEXT true
*EUC、JIS、Unicode、または ISO ラテン文字セットにコード化された人間の言語 (MIME-TYPE text/plain; charset=XXX)	IS_TEXT true
カレンダ・アポイント (MIME_TYPE text/plain)	IS_TEXT false

表 9–3　IS_TEXT 属性の例　続く

説明と **MIME_TYPE** 属性	**IS_TEXT** 値
ハイパーテキスト・マークアップ言語 (HTML) (MIME_TYPE text/html)	IS_TEXT true
PostScript (MIME_TYPE application/postscript)	IS_TEXT false
C プログラム・ソース (C_SRC) (MIME_TYPE text/plain)	IS_TEXT true
ビットマップとピックスマップ (XBM と XPM) (MIME_TYPE text/plain)	IS_TEXT false
デスクトップ・アプリケーション・ビルド・サービスのためのプロジェクトまたはモジュール・ファイル (MIME_TYPE text/plain)	IS_TEXT false
シェル・スクリプト (MIME_TYPE text/plain)	IS_TEXT false
uuencode(1) によって生成されたコード化テキスト (MIME_TYPE text/plain)	IS_TEXT false
*MIME_TYPE text/plain	IS_TEXT false

データ型属性の詳細は、dtdtsfile(4) のマニュアル・ページを参照してください。

データ型関数

データ・オブジェクトの属性を調べるには、まずオブジェクトの型を判断し、その型の適切な属性値を求めなければなりません。データベースにデータ情報を問い合わせるための関数を表 9–4 に示します。セクション 3 にこれらの関数のマニュアル・ページがあります。詳細は、該当するマニュアル・ページを参照してください。

表 9–4　データ型データベース問い合わせ関数

関数	説明
DtDtsBufferToAttributeList()	指定バッファのデータ属性のリストを検索します。
DtDtsBufferToAttributeValue()	指定バッファのデータ属性を検索します。
DtDtsBufferToDataType()	指定バッファのデータ型名を検索します。
DtDtsDataToDataType()	指定データ・セットのデータ型を検索します。
DtDtsDataTypeIsAction()	結果として保存されたディレクトリのデータ型を返します。
DtDtsDataTypeNames()	使用可能なデータ型のリストを検索します。
DtDtsDataTypeToAttributeList()	指定データ属性名の属性リストを検索します。
DtDtsDataTypeToAttributeValue()	指定データ属性名の属性値を検索します。
DtDtsFileToAttributeList()	指定ファイルのデータ属性のリストを検索します。
DtDtsFileToAttributeValue()	指定ファイルのデータ属性値を検索します。
DtDtsFileToDataType()	指定ファイルのデータ型を検索します。
DtDtsFindAttribute()	属性 name が value に一致するデータ型のリストを検索します。
DtDtsFreeAttributeList()	指定属性リストのメモリを解放します。
DtDtsFreeAttributeValue()	指定属性値のメモリを解放します。
DtDtsFreeDataType()	指定データ型名のアプリケーション・メモリを解放します。

表 9–4 データ型データベース問い合わせ関数 続く

関数	説明
DtDtsFreeDataTypeNames()	DtDtsDataTypeNames() または DtDtsFindAttribute() を呼び出して作成されたメモリを解放します。
DtDtsIsTrue()	文字列を論理値に変換する簡易関数
DtDtsRelease()	一般的には再読み込みの準備として、データ型データベース情報の読み込みを解除します。
DtDtsSetDataType()	指定されたディレクトリのデータ型を設定します。
DtsLoadDataTypes()	データ型関数のためにデータベース・フィールドを初期化し、読み込みます。アクションまたはアクション型を使用する必要がなく、パフォーマンスを向上させたい場合は、DtDbLoad() の代わりに使用します。アクションを使用する必要がある場合は DtDbLoad() を使用します。

データ型を検査して属性を検索するには、簡易、中間、拡張の 3 つの方法があります。

簡易データ型検査

データ型を検査するための最も簡単な方法は、次の関数を使用することです。

- DtDtsFileToAttributeList()
- DtDtsFileToAttributeValue()

これらの関数を使用すると、ファイルの型が検査され、単一の属性またはリスト全体が検索されます。システム・コールが行われ、データ型の検査と属性の検索が行われます。次の関数は、中間データ型検査関数を呼び出します。

- DtDtsBufferToAttributeList()
- DtDtsBufferToAttributeValue()

バッファは、読み取り権／書き込み権を持つ通常ファイルに一致するモードを持つと想定されます。読み取り専用バッファの型の検査については、138ページの「拡張データ型検査」を参照してください。

中間データ型検査

データの型を検査して属性を検索する場合、プロセスのデータ型検査部分は、パフォーマンスの点で最もコストがかかります。データ型の検査を 2 番目の方法で行うと、データ型検査のための関数と属性検索のための関数を切り離すことによって、パフォーマンスを改善できます。中間データ型検査には、次の関数を使用します。

- `DtDtsBufferToDataType()`
- `DtDtsFileToDataType()`
- `DtDtsDataTypeToAttributeList()`
- `DtDtsDataTypeToAttributeValue()`

アプリケーションが複数の属性値を問い合わせる場合には、これらの関数を使用します。これらの関数を使用すると、オブジェクトの型が検査され、その型を使用して属性リストから 1 つ以上の属性を検索します。

データ型検査と属性の検索を行うには、中間データ型関数を使用するようにしてください。これらの関数は、拡張データ型関数を呼び出し、バッファについて簡易データ型検査と同様に想定します。

拡張データ型検査

拡張データ型検査では、システム・コール、データ型、さらには属性検索も別々に行われます。拡張データ型検査では、あらかじめ初期化されてデータ型関数の一部としては含まれない既存のシステム・コールからのデータを使用するので、コード化が複雑になります。拡張データ型検査には、次の関数を使用してください。

`DtDtsDataToDataType()`

読み取り専用バッファの型を検査するには、`st_mode` フィールドが `S_IFREG | S_IROTH |S_IRGRP | S_IRUSR` に設定された `stat` 構造体が渡されなければなりません。

アクションであるデータ型 (DtDtsDataTypeIsAction)

データベースが読み込まれるとアクションの検査ができるようになるため、データベースの各アクションに対して合成データ型が生成されます。これらのデータ型は、次の 2 つの追加の属性を持つことができます。

- IS_ACTION は、このデータ型がアクションであることをユーザに知らせる文字列論理値です。IS_ACTION に文字列 true (大文字と小文字の区別はありません) が設定されている場合、データはアクションです。
- IS_SYNTHETIC は、このデータ型が ACTION テーブルのエントリから生成されたことをユーザに知らせる文字列論理値です。IS_SYNTHETIC に true が設定されている場合、データ型は生成されています。

ドロップ領域としてのオブジェクトの登録

アプリケーションがデータ型を定義する場合は、次の手順に従ってプログラマが意図したドラッグ & ドロップ動作のすべてが提供されているか確認してください。

1. アプリケーションの中で、データ型を定義する必要があるかどうかを指定します。
2. 定義する各データ型について、関連するオブジェクトをドロップ領域にするかどうかを指定します。
3. ドロップ領域として登録する各オブジェクトについて、どの操作 (移動、コピー、またはリンク) を定義するかを指定します。
4. 各オブジェクトに対して有効なドロップ操作について、適切なドロップ・アクションを定義します (MOVE_TO_ACTION、COPY_TO_ACTION、および LINK_TO_ACTION 属性を設定してください)。

アプリケーションがデータ・オブジェクトのアイコンを表示する場合、それらのアイコンをドロップ領域としてサポートしなければならないこともあります。その場合、MOVE_TO_ACTION、COPY_TO_ACTION、または LINK_TO_ACTION 属性を問い合わせて、それらのデータ・オブジェクトのドロップ動作を指定する必要があります。対応する属性値が NULL でない場合だけ、オブジェクトはドロップ操作をサポートしなければなりません。3 つの属性すべてが NULL の値を持つ場合、オブ

ジェクトはドロップ領域として登録されません。データ型が定義されているオブジェクトの属性を最低 1 つでも設定すると、アプリケーションはそのオブジェクトをドロップ領域として登録できます。

ユーザがオブジェクトをドロップ領域にドラッグすると、アプリケーションはドロップを行うためにどのジェスチャ (すなわち、どのドラッグ操作) が使用されたかを判断します。ドラッグ操作とドロップ領域のデータ型に基づいて、アプリケーションはデータ型データベースからドロップ属性を検索します。次に、DtActionInvoke を呼び出して、次の 2 つの規則によってパラメータを判断します。

1. ユーザがオブジェクト **A** と **B** をオブジェクト **C** 上にドロップした場合は、**C**、**A**、**B** を args として DtActionInvoke を呼び出します。action は、**C** の MOVE_TO_ACTION、COPY_TO_ACTION、LINK_TO_ACTION のいずれかの値です。オブジェクト **C** がアクションの場合、args リストは **C** を含みません。また、action は **C** です。

ファイル・マネージャとそのディレクトリおよびフォルダ・オブジェクトは、デスクトップが移動、コピー、およびリンクされたドロップ属性を使用する方法を示す例となります。ユーザは、オブジェクト (ファイル) をディレクトリ・フォルダへドラッグ & ドロップできます。ファイル・マネージャは、フォルダ・オブジェクトに対して、MOVE_TO_ACTION、COPY_TO_ACTION、および LINK_TO_ACTION アクションを定義します。これらのアクションは、適切なファイル・システムの移動、コピー、およびリンクのためのシステム関数を実行します。

MOVE_TO_ACTION、COPY_TO_ACTION、および LINK_TO_ACTION 属性の定義の例については、/usr/dt/appconfig/types/C/dtfile.dt を参照してください。ドラッグ & ドロップの使用方法の詳細は、第 5 章を参照してください。

データ型データベースの使用例

この節では、データ型検査のコード例を示します。このコード例は、/usr/dt/examples/dtdts/datatyping.c にあります。このサンプル・コードは、渡された各ファイルのデータ型、アイコン名、およびサポートされるアクションを示します。dtaction クライアントを使用して、サポートされているアクションをファイルで実行することもできます。datatyping の使い方は、次のとおりです。

```
datatyping file1 [file2 ...]
#include <Xm/Form.h>
#include <Xm/Text.h>
#include <Dt/Dts.h>

#define ApplicationClass "DtDatatyping"

static Widget text;

static void DisplayTypeInfo(int, char**);

int main(int argc, char **argv)
{
     XtAppContext appContext;
     Widget toplevel, form;
     Arg args[20];
     int n;

     toplevel = XtAppInitialize(&appContext, ApplicationClass,
    NULL, 0,
          argc, argv, NULL, NULL, 0);

     if (argc == 1) {
         printf("%s: No files specified.\n", argv[0]);
         exit(1);
     }

     form = XmCreateForm(toplevel, "form", NULL, 0);
     XtManageChild(form);
     n = 0;
     XtSetArg(args[n], XmNleftAttachment, XmATTACH_FORM); n++;
     XtSetArg(args[n], XmNrightAttachment, XmATTACH_FORM); n++;
     XtSetArg(args[n], XmNtopAttachment, XmATTACH_FORM); n++;
     XtSetArg(args[n], XmNbottomAttachment, XmATTACH_FORM); n++;
     XtSetArg(args[n], XmNeditable, False); n++;
     XtSetArg(args[n], XmNeditMode, XmMULTI_LINE_EDIT); n++;
     XtSetArg(args[n], XmNrows, 25); n++;
     XtSetArg(args[n], XmNcolumns, 90); n++;
     text = XmCreateScrolledText(form, "text", args, n);
     XtManageChild(text);

     XtRealizeWidget(toplevel);
    if (DtAppInitialize(appContext, XtDisplay(toplevel), toplevel, argv[0],
                                                ApplicationClass) == False) {
         printf("%s: Couldn't initialize Dt\n", argv[0]);
         exit(1);
     }

     DtDbLoad();

     DisplayTypeInfo(argc, argv);

     XtAppMainLoop(appContext);
}

static void DisplayTypeInfo(int argc, char **argv)
{
     char *file;
     char *datatype;
     char *icon;
```

```
    char *actions;
    char str[100];
    int i;

    sprintf(str, "%-30s\t%-10s\t%-8s\t%-20s\n",
                "File",
                "DataType",
                "Icon",
                "Actions");
    XmTextInsert(text, XmTextGetLastPosition(text), str);

    sprintf(str, "%-30s\t%-10s\t%-8s\t%-20s\n",
                "--------------------",
                "---------",
                "----",
                "--------");
    XmTextInsert(text, XmTextGetLastPosition(text), str);

     for(i=1; i < argc; i++) {
        char *file = argv[i];

        /* find out the Dts data type */
        datatype = DtDtsFileToDataType(file);

        if(datatype) {
            /* find the icon attribute for the data type */
            icon = DtDtsDataTypeToAttributeValue(datatype,
       DtDTS_DA_ICON, file);
         }

    /*  Directly find the action attribute for a file */

          actions = DtDtsFileToAttributeValue(file,
     DtDTS_DA_ACTION_LIST);
          sprintf(str, "%-30s\t%-10s\t%-8s\t%s\n",
                         file,
                         datatype?datatype:"unknown",
                        icon?icon:"unknown",
                         actions?actions:"unknown");
        XmTextInsert(text, XmTextGetLastPosition(text), str);

        /* Free the space allocated by Dts */

        DtDtsFreeAttributeValue(icon);
        DtDtsFreeAttributeValue(actions);
        DtDtsFreeDataType(datatype);
}
```

第 10 章

カレンダとの統合

カレンダのアプリケーション・プログラム・インタフェース (API) は、ネットワーク環境でカレンダ・データにアクセスし、管理するためのプログラム的な方法を提供します。API は、項目の挿入、削除、変更だけでなく、ブラウズおよび検索機能もサポートします。また、カレンダ管理関数をサポートします。

カレンダ API は、X.400 Application Programming Interface Association (XAPIA) の Calendaring and Scheduling API (CSA API) を実装しています。CSA API は、カレンダが有効なアプリケーションからカレンダおよびスケジュール・サービスのさまざまな機能へのアクセスを可能にする高水準の関数のセットを定義しています。最新の XAPIA 仕様の詳細は、X.400 API Association (800 El Camino Real, Mountain View, California 94043) に問い合わせてください。

この章では、次の節でカレンダ API を説明します。

ライブラリとヘッダ・ファイル

カレンダ API を使用するには、`libcsa` ライブラリをリンクする必要があります。ヘッダ・ファイルは、`csa/csa.h` です。

デモ・プログラム

カレンダ API の使用例を示すデモ・プログラムが、`/usr/dt/examples/dtcalendar` にあります。

カレンダ **API** の使い方

▼ カレンダと統合するには

カレンダ API は、ネットワーク環境でカレンダ・データにアクセスし、管理する方法を提供します。

1. アプリケーションに `csa/csa.h` を組み込みます。
2. カレンダ **API** を使用して、アプリケーションの中で使用するカレンダ操作を組み込みます。
3. `libcsa` とリンクします。

CSA API の概要

CSA インタフェースは、カレンダおよびスケジュール・サービスへの共通インタフェースを可能にします。CSA 実装のそれぞれについて、CSA によって与えられる表示と機能は、基本のカレンダ・サービスの表示と機能にマップされなければなりません。インタフェースは、実際のカレンダおよびスケジュールの実装に依存しないように設計されています。また、インタフェースは、カレンダ・サービスが使用するオペレーティング・システムと基本のハードウェアに依存しないように設計されています。

提供される関数呼び出しの数は、最小限のものです。一組の関数で複数の種類のカレンダ項目を管理します。

C の命名規則

表 10–1 に示すように、C インタフェースの要素の識別子は、要素の属性名とそれに関連するデータ型に由来します。属性名には、テーブルの 2 番目の欄の文字列が接頭辞として付けられます。英字は、3 番目の欄の大文字または小文字に変換されます。

表 10–1 C 命名規則の由来

要素の種類	接頭辞	大文字／小文字
データ型	CSA_	小文字
データの値	CSA_	大文字
関数	csa_	小文字
関数の引き数	なし	小文字
関数の結果	なし	小文字
定数	CSA_	大文字
エラー	CSA_E_	大文字

表 10–1　C 命名規則の由来　続く

要素の種類	接頭辞	大文字／小文字
マクロ	CSA_	大文字
拡張セットのために確保	CSA_XS_	大文字／小文字
拡張のために確保	CSA_X_	大文字／小文字
処理系作成者が使用するために確保	CSAP	大文字／小文字
ベンダ関数拡張のために確保	csa_x	小文字
構造体のタグ	CSA_TAG_	大文字

接頭辞 CSAP (大文字／小文字) が付いている要素は、CSA サービスの実装の作成者が内部専用として使用するために確保されています。CSA インタフェースによって書かれたプログラムが直接使用するためのものではありません。

接頭辞 CSA_XS_、CSA_X_ (大文字／小文字)、および csa_x は、ベンダまたはグループによるインタフェースの拡張のために確保されています。仕様では、これらのインタフェース拡張は、基本関数セットの拡張として定義されています。

定数データ値の場合、定数データ値のデータ構造体または関数を示すために、通常、追加の文字列が CSA_ に追加されます。

機能のアーキテクチャ

本節では、CSA API をサポートしているサービスの機能のアーキテクチャを説明します。抽象実装モデル、抽象データ・モデル、および機能の概要を示します。

実装モデル

CSA API の適用範囲が理解できるように、抽象実装モデルが用意されています。

CSA インタフェースは、カレンダが使用可能なアプリケーションとカレンダ・サービスの間に定義されます。このインタフェースの機能はすべて、カレンダ・サービスに依存しないように設計されています。ただし、この API では、拡張の使用によって実行される共通関数のプロトコル固有の拡張は許されています。詳細は、153ページの「拡張」を参照してください。カレンダが使用可能なアプリケーションとカレンダ・サービスの CSA インタフェースの関係を図 10–1 に示します。

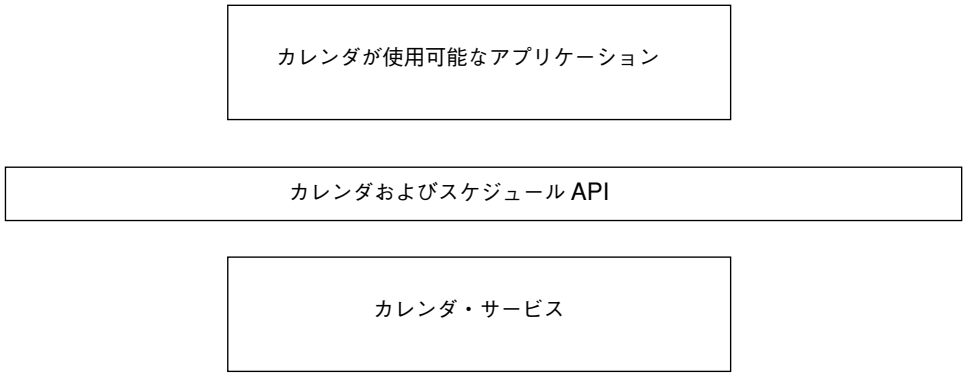

図 10–1　カレンダおよびスケジュール API の位置付け

CSA インタフェースのモデルは、管理、カレンダ管理、および項目管理という 3 つのコンポーネントに分けることができます。これらのコンポーネントを図 10–2 に示します。

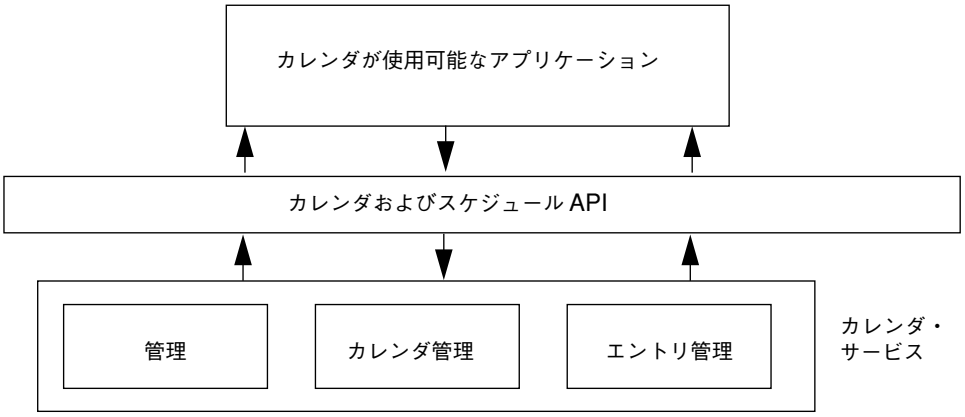

図 10–2　カレンダおよびスケジュール API のコンポーネント

カレンダ・サービスへのアクセスは、カレンダ・セッションを通して確立されます。セッションは、カレンダ・サービスへの有効な接続のために用意され、サービスによって保持されるカレンダ情報の整合性の確保を支援します。カレンダが使用可能なアプリケーションは、カレンダ・サービス内の個々のカレンダにログインして、有効なセッションまたは接続を確立します。セッションは、カレンダが使用可能なアプリケーションがカレンダからログアウトすることによって終了します。

カレンダ・サービスは、1 つ以上のカレンダを保持します。カレンダ・サービスは、これらのカレンダに対して、いくつかのレベルの管理サポートを提供します。カレンダが使用可能なアプリケーションは、特定のカレンダ・サービスによって保持されるカレンダのリストにアクセスできます。さらに、カレンダ・サービスにより、実装固有の永続的形式にカレンダ情報を保管したり復元したりできます。カレンダ・サービスが複数のカレンダの保持をサポートする場合には、カレンダの作成と削除のためのサポート関数が定義されます。また、カレンダの特性を管理するための関数が定義されます。

CSA インタフェースのほとんどの関数は、個々のカレンダ項目を管理します。カレンダ項目は、イベント、予定、またはメモです。項目は、特定のカレンダへの追加、削除、更新、および読み取りができます。カレンダが使用可能なアプリケーションは、カレンダ項目に通知方法を追加できます。

データ・モデル

CSA インタフェースは、カレンダ・サービスによって保持されるカレンダ情報の概念上のバックエンドの記憶領域へのアクセス方法です。共通データ・モデルは、カ

レンダ・サービスによって保持されるカレンダ情報のコンポーネントを視覚化する際に役に立ちます。

カレンダ・エンティティ

データ・モデルは、カレンダ・エンティティの概念に基づきます。カレンダは、管理カレンダ属性とカレンダ項目の名前付きコレクションによって表されます。カレンダは、個々のユーザによって所有されます。ユーザは、個人、グループ、またはリソースを表します。

カレンダ属性は、カレンダに関する共通、実装固有、またはアプリケーション固有の管理特性を表す名前付きの値のセットです。たとえば、タイムゾーン、名前、所有者、およびカレンダへのアクセスの権利を、個々のカレンダ属性の中で指定できます。

カレンダ項目は、カレンダの主要なコンポーネントです。カレンダ項目の 3 つのクラスは、次のとおりです。

- イベント
- 予定
- メモ

カレンダ項目は、固有な名前を付けられた項目属性のコレクションによって表されます。項目属性は、カレンダ項目の共通、実装固有、またはアプリケーション固有の特性を表す名前付きの値のセットです。たとえばイベントには、開始と終了の日付と時間、説明、およびサブタイプを指定できます。予定には、作成日、期限、優先順位、およびステータスを指定できます。メモには、作成日とテキスト内容または説明を入れることができます。

カレンダ属性と項目属性は、名前、型、値の 3 つの組から成ります。仕様によって定義されている共通属性を拡張できます。実装によって、固有の属性を定義できます。また、アプリケーションでアプリケーション固有の属性を定義するための機能を提供するものもあります。共通デスクトップ環境では、アプリケーション定義の属性をサポートします。

アクセス権

個々のユーザがカレンダにアクセスできるかどうかは、そのユーザに与えられるアクセス権によって制御されます。アクセス権は、カレンダのユーザと対になっています。CSA では、ユーザは、個人、グループ、またはリソースです。共通デスク

トップ環境では、個々のユーザだけをサポートします。アクセス権は、アクセス・リストで保持されます。アクセス・リストは、特定のカレンダ属性です。アクセス権は、個別に制御され、それを累積することによって、カレンダとその項目に対するユーザのアクセスの範囲を定義できます。アクセス権は、次のアクセスの役割の観点から指定できます。

- カレンダの所有者
- カレンダ内の特定の項目の主催者
- カレンダ内の特定の項目のスポンサー

所有者の役割を与えられたユーザは、カレンダの所有者ができることであれば、カレンダまたはカレンダ項目に対して何でも実行できます。すなわち、カレンダの削除、カレンダ属性の表示、挿入、変更、カレンダ項目の追加と削除、項目属性の表示、挿入、および変更を実行できます。

主催者の役割を与えられたユーザは、そのユーザが主催者として指定されたカレンダ項目に対して、項目の削除、または項目属性の表示や変更を実行できます。デフォルトでは、項目を作成したカレンダ・ユーザが主催者です。

スポンサーの役割を与えられたユーザは、そのユーザがスポンサーとして指定されたカレンダ項目に対して、項目の削除、または項目属性の表示や変更を実行できます。スポンサーは、カレンダ項目を実質的に所有するカレンダ・ユーザです。

これらの役割に加えて、アクセス権の設定によって、公用、半私用、私用の分類に応じて、空き時間の検索へのアクセス、カレンダ属性の表示、挿入、変更、あるいは項目の表示、挿入、変更を制限できます。項目の分類は、アクセスできるかどうかの二次フィルタとして機能します。

機能の概要

CSA インタフェースは、主に 3 種類の作業をサポートします。

- 管理
- カレンダ管理
- エントリ管理

管理

CSA 関数呼び出しの大部分は、カレンダ・セッションの中で発生します。カレンダ・セッションは、カレンダが使用可能なアプリケーションとカレンダ・サービス

によって保持された特定のカレンダとの間の論理的な接続です。セッションは、csa_logon() 関数の呼び出しで確立され、csa_logoff() 関数の呼び出しで終了します。セッションの状況は、セッション・ハンドルによって表されます。このハンドルは、1 つのカレンダ・セッションを他のセッションと見分けるためのトークンを各 CSA 関数の中で提供します。csa_logon() 関数は、また、カレンダ・サービスに対してユーザを認証し、セッション属性を設定します。現時点では、アプリケーション間でのカレンダ・セッションの共有はサポートされていません。

csa_list_calendars() 関数は、特定のカレンダ・サービスによって管理されるカレンダ名をリストするために使用されます。

csa_query_configuration()関数は、現在のカレンダ・サービスの構成に関する情報をリストするために使用されます。この情報は、文字セット、テキスト文字列の行終了文字、デフォルトのサービス名、指定されたカレンダ・サービスのデフォルトの認証ユーザ識別子、ユーザ識別子を認証するためにパスワードが必要かどうかを示すインジケータ、ユーザ・インタフェース・ダイアログの共通拡張がサポートされるかどうかを示すインジケータ、および実装によってサポートされる CSA 仕様などです。

CSA の実装は、サービスによって返されるカレンダ・オブジェクトおよび属性のためのメモリの管理をサポートします。csa_free() 関数は、このメモリが不要になったときに、解放するために使用されます。カレンダ・サービスによって割り当てられ、管理されるメモリを解放するのは、アプリケーションの責任です。

カレンダ管理

CSA インタフェースは、いくつかのカレンダ管理関数を提供します。共通デスクトップ環境では、1 つのカレンダ・サービスにつき複数のカレンダをサポートします。カレンダが使用可能なアプリケーションは、カレンダを追加したり削除したりできます。csa_delete_calendar() 関数は、カレンダを削除するために使用されます。csa_add_calendar() 関数は、サービスに新しいカレンダを追加するために使用されます。

アプリケーションは、また、csa_list_calendar_attributes()、csa_read_calendar_attributes()、および csa_update_calendar_attributes() 関数を使用して、カレンダ属性のリスト、読み取り、および更新を実行できます。アプリケーションは、カレンダ・ログイン、カレンダの削除、カレンダ属性の更新、新しいカレンダ項目の追加、カレンダ項目の削除、およびカレンダ項目の更新について通知を受けるためのコールバック関数を登録できます。コールバック関数は、カレンダ・セッションの継続中

だけ登録されます。この情報は、一部のカレンダ管理アプリケーションにとっては貴重なものです。

エントリ管理

CSA インタフェースは、カレンダ項目を管理するための強力な関数のセットを備えています。カレンダ・セッション中のカレンダ項目の状況は、項目ハンドルによって保持されます。このハンドルは、1 つのカレンダ項目を他の項目と見分けるためのトークンを CSA 関数の中で提供します。項目ハンドルは、`csa_add_entry()` と`csa_list_entries()` 関数によって返されます。項目ハンドルは、カレンダ・セッションの継続期間、あるいは項目が削除または更新されるまで有効です。`csa_free()` の呼び出しによって解放されると、項目ハンドルは無効になります。

`csa_add_entry()` 関数は、カレンダに新しい項目を追加するために使用されます。`csa_delete_entry()` 関数は、カレンダの中の項目を削除するために使用されます。`csa_list_entries()` 関数は、項目属性基準の特定のセットと一致するカレンダ項目を列挙するために使用されます。`csa_read_entry_attributes()` 関数は、特定のカレンダ項目に関連するすべてまたは一組の項目属性値を取り出すために使用されます。

カレンダに項目を追加するには、カレンダが使用可能なアプリケーションは、まず `csa_logon()` 関数を使用して、カレンダ・サービスとのセッションを確立しなければなりません。次に、アプリケーションは、`csa_add_entry()` 関数を新しい項目を指定するために実行します。カレンダが使用可能なアプリケーションは、`csa_add_entry()` 関数の中で使われる属性を組み立てる責任があります。セッションの終了には、`csa_logoff()` 関数が使用されます。

個々のカレンダ項目の中の項目属性は、`csa_list_entry_attributes()` 関数で列挙できます。`csa_read_entry_attributes()` 関数を使用すると、1 つ以上の属性の値を読み取ることができます。個々の項目属性は、`csa_update_entry_attributes()` 関数で変更できます。

カレンダ情報を検索するために CSA の実装によって割り当てられたメモリは、関連するメモリ・ポインタを `csa_free()` 関数に渡すことによって解放されます。

再帰的活動に関連するカレンダ項目もあります。`csa_list_entry_sequence()` 関数を使用すると、他の再帰的カレンダ項目を列挙できます。この関数は、再帰的項目の項目ハンドルのリストを返します。

CDE カレンダ・サーバは、カレンダ項目に関連付けられるアラームまたは通知方法のサポートを提供します。通知方法は、端末のスピーカからの音声による通知、端末画面の点滅による通知、カレンダ・ユーザへのメール送信による通知、端末画面にポップアップを表示することによる通知などの形を取ることができます。カレンダ・サービスは通知方法を管理しますが、通知情報を検索し、情報に対処するのはカレンダ・アプリケーションの責任です。`csa_read_next_reminder()` 関数は、次のスケジュール済みの通知に関する情報を読み込むために使用されます。

拡張

CSA 仕様で定義されている大半のデータ構造と関数は拡張できます。拡張は、データ構造にフィールドを追加したり、関数呼び出しにパラメータを追加したりするために行われます。これらの拡張のための標準的な汎用データ構造が定義されています。それは、拡張を識別する項目コード、拡張データまたはデータ自体の長さを保持する項目データ、拡張値が格納されている場所を示す項目参照と、関連する項目の格納領域がない場合には `NULL`、および拡張のフラグから成ります。

関数呼び出しにパラメータを追加するような拡張を、入力または出力時に実行できます。すなわち、拡張は、アプリケーションから CSA サービスへの入力パラメータとして渡すことができ、または、CSA サービスからアプリケーションへの出力パラメータとして渡すこともできます。拡張が入力パラメータの場合には、アプリケーションは、拡張構造体と、その拡張に関連するその他の構造体のためのメモリを割り当てます。拡張が出力パラメータの場合には、CSA サービスは必要に応じて、拡張の結果のための記憶領域を割り当てます。この場合、アプリケーションは、割り当てられた記憶領域を `csa_free()` 呼び出しによって解放しなければなりません。

サポートされていない拡張が要求された場合には、`CSA_E_UNSUPPORTED_FUNCTION_EXT` が返されます。

共通デスクトップ環境 (CDE) の実装

CSA API の CDE 実装は、CDE カレンダ・サーバへのアクセスを可能にするライブラリです。ライブラリとサーバとの通信には、ONC の RPC が使用されます。CDE 実装におけるカレンダ・サーバは、カレンダ・プロトコル・バージョン 2 から 5、およびデータ・バージョン 3 と 4 をサポートするバージョン 5 です。カレンダ・プロトコルのバージョン 2 から 4 とデータ・バージョン 3 は、OpenWindows カレンダ・マネージャへの下位互換を確保するためのものです。カレンダ・プロト

コル・バージョン 5 とデータ・バージョン 4 は CSA インタフェースとデータの拡張性をサポートします。

表 10–2 サポートされるサーバのバージョンとデータのバージョン

サーバのバージョン	データのバージョン
2	1
3	2
4	3
5	3、4

サポートされるアクセス・モデル

2 つのアクセス・モデルがカレンダ API によってサポートされています。XAPIA CSA 仕様において指定されているアクセス・モデルは、データ・バージョン 4 のためだけにサポートされています。OpenWindows カレンダ・マネージャのアクセス・モデルは、データ・バージョン 1 から 3 までのためにサポートされています。OpenWindows のカレンダ・マネージャ・アクセス・モデルでは、カレンダのアクセス許可は、アクセス権を指定するアクセス・リストにより制御されます。次の 3 種類のアクセス権が定義されています。

`CSA_X_DT_BROWSE_ACCESS` (ユーザはカレンダのエントリをリストして、読み取ることができる)

`CSA_X_DT_INSERT_ACCESS` (ユーザはカレンダのエントリを挿入できる)

`CSA_X_DT_DELETE_ACCESS` (ユーザはカレンダのエントリを削除できる)

カレンダ API はすべてのバージョンのカレンダへのアクセスを可能にするので、プログラマは、データ・バージョンに対応する正しいアクセス・モデルを使用して、アクセス・リストに含まれるアクセス権を解釈しなければなりません。

カレンダが作成されるとき、アクセス・リストを指定しないかぎり、デフォルトのアクセス・リストにユーザ名として world という 1 つのエントリが含まれます。world のアクセス権では、公開エントリをブラウズできます。world というユーザ名は、すべてのユーザを意味する特別の名前です。

デフォルトでは、カレンダの所有者と同じユーザ名を持つユーザは、任意のマシンから、所有者のアクセス権でカレンダにアクセスできます。さらに厳しいアクセス制御をするには、owner-user-name@host という書式の名前をカレンダのアクセス・リストに追加できます。このようなエントリをアクセス・リストに追加するときは、対応するアクセ
ス権は、データ・バージョン 4 では CSA_OWNER_RIGHTS、データ・バージョン 3 では (CSA_X_DT_BROWSE_ACCESS|CSA_X_DT_INSERT_ACCESS|CSA_X_DT_DELETE_ACCESS) です。このようなエントリをアクセス・リストに追加した後は、指定されたホストからのユーザだけが所有者のすべての権利でカレンダにアクセスできます。

データ構造

表 10–3 に、CSA データ構造をリストします。詳細は、関連するマニュアル・ページを参照してください。

表 10–3 CSA データ構造

データ型の名前	説明
Access List	カレンダ・ユーザのアクセスの権利構造体のリスト
Attendee List	出席者構造体のリスト
Attribute	属性構造体
Attribute Reference	属性参照構造体
Boolean	論理的な True または False を示す値
Buffer	データ項目のポインタ
Calendar User	カレンダ・ユーザ構造体
Callback Data Structures	コールバック・データ構造体
Date and Time	日付と時間の指定
Date and Time List	日付と時間の値のリスト
Date and Time Range	日付と時間の範囲

表 10–3　CSA データ構造　続く

データ型の名前	説明
Entry Handle	カレンダ項目のハンドル
Enumerated	計算の値を含むデータ型
Extension	拡張構造体
Flags	ビート・マスクのコンテナ
Free Time	空き時間構造体
Opaque Data	不透明データ構造体
Reminder	通知方法構造体
Reminder Reference	通知方法参照構造体
Return Code	関数が成功したこと、または失敗した理由を示す戻り値
Service Reference	サービス参照構造体
Session Handle	カレンダ・セッションのハンドル
String	文字列ポインタ
Time Duration	継続時間

カレンダ属性

表 10–4 に、共通デスクトップ環境でサポートされるカレンダ属性をリストします。詳細は、関連するマニュアル・ページを参照してください。カレンダ属性のリストは、拡張命名規則による拡張が可能です。

表 10–4 CSA カレンダ属性

属性名	記号名	サーバのバージョン	データのバージョン	読み取り専用
Access List	CSA_CAL_ATTR_ACCESS_LIST_	2-5	1-4	×
Calendar Name	CSA_CAL_ATTR_CALENDAR_NAME	2-5	1-4	○ *
Calendar Owner	CSA_CAL_ATTR_CALENDAR_OWNER	2-5	1-4	○ *
Calendar Size	CSA_CAL_ATTR_CALENDAR_SIZE	5	3,4	○
Character Set	CSA_CAL_ATTR_CHARACTER_SET	5	4	○
Data Version**	CSA_X_DT_CAL_ATTR_DATA_VERSION	2-5	1-4	○
Date Created	CSA_CAL_ATTR_DATE_CREATED	5	4	○
Number Entries	CSA_CAL_ATTR_NUMBER_ENTRIES	2-5	1-4	○
Product Identifier	CSA_CAL_ATTR_PRODUCT_IDENTIFIER	2-5	1-4	○
Server Version**	CSA_X_DT_CAL_ATTR_SERVER_VERSION	2-5	1-4	○
Time Zone	CSA_CAL_ATTR_TIME_ZONE	5	4	○
Version	CSA_CAL_ATTR_VERSION	2-5	1-4	○

* カレンダ作成時に指定し、その後は読み取り専用になります。

** CDE のみ

次のカレンダ属性はサポートされません。

```
CSA_CAL_ATTR_COUNTRY
CSA_CAL_ATTR_LANGUAGE
```

(続く)

```
CSA_CAL_ATTR_WORK_SCHEDULE
```

次の節では、表 10–4 にリストしたカレンダ属性について、追加の情報を提供します。

- Access List

  新しいカレンダが追加されるときにアクセス・リストが指定されなかった場合には、デフォルトのアクセス・リストには特殊ユーザ world が指定され、それに対応するアクセス権は CSA_VIEW_PUBLIC_ENTRIES になります。これは、公用のカレンダ・エントリのリストと読み取りのアクセス権を与えます。特殊ユーザ world には、すべてのユーザが含まれます。

- Calendar Name

  カレンダ名は、csa_add_calendar() によってカレンダが作成されるときに指定されます。読み取り専用であり、カレンダの作成後に変更できません。

- Calendar Owner

  カレンダ所有者は、csa_add_calendar() を呼び出してカレンダを作成するアプリケーションを実行しているユーザに設定されます。読み取り専用で、カレンダの作成後に変更できません。

- Character Set

  この値の読み取り設定には、CDE 共通ロケール名が使用されます。

CDE 定義済みカレンダ属性は、次のとおりです。

- Server Version

  この読み取り専用の属性は、カレンダを管理しているサーバのバージョン番号を示します。この属性は、CSA_VALUE_UINT32 型の属性です。

- Data Version

  この読み取り専用の属性は、カレンダのデータ・バージョンを示します。この属性は、CSA_VALUE_UINT32 型の属性です。

項目属性

表 10–5 に、共通デスクトップ環境でサポートされる項目属性をリストします。詳細は、関連するマニュアル・ページを参照してください。項目属性のリストは、拡張命名規則による拡張が可能です。

表 10–5 CSA 項目属性

属性名	記号名	サーバのバージョン	データのバージョン	読み取り専用
Audio Reminder	CSA_ENTRY_ATTR_AUDIO_REMINDER	2-5	1-4	×
Character Set*	CSA_X_DT_ENTRY_ATTR_CHARACTER_SET	5	4	×
Classification	CSA_ENTRY_ATTR_CLASSIFICATION	5	2-4	×
Date Completed	CSA_ENTRY_ATTR_DATE_COMPLETED	5	4	×
Date Created	CSA_ENTRY_ATTR_DATE_CREATED	5	4	○
Description	CSA_ENTRY_ATTR_DESCRIPTION	5	4	×
Due Date	CSA_ENTRY_ATTR_DUE_DATE	5	4	×
End Date	CSA_ENTRY_ATTR_END_DATE	2-5	1-4	×
Exception Dates	CSA_ENTRY_ATTR_EXCEPTION_DATES	5	4	×
Flashing Reminder	CSA_ENTRY_ATTR_FLASHING_REMINDER	2-5	1-4	×
Last Update	CSA_ENTRY_ATTR_LAST_UPDATE	5	4	○
Mail Reminder	CSA_ENTRY_ATTR_MAIL_REMINDER	2-5	1-4	×
Number Recurrences	CSA_ENTRY_ATTR_NUMBER_RECURRENCES	5	4	○
Organizer	CSA_ENTRY_ATTR_ORGANIZER	2-5	1-4	○

表 10–5　CSA 項目属性　続く

属性名	記号名	サーバのバージョン	データのバージョン	読み取り専用
Popup Reminder	CSA_ENTRY_ATTR_POPUP_REMINDER	2-5	1-4	×
Priority	CSA_ENTRY_ATTR_PRIORITY	5	4	×
Recurrence Rule	CSA_ENTRY_ATTR_RECURRENCE_RULE	5	4	×
Reference Identifier	CSA_ENTRY_ATTR_REFERENCE_IDENTIFIER	2-5	1-4	○
Repeat Interval*	CSA_X_ENTRY_ATTR_REPEAT_INTERVAL	2-5	1-4	**
Repeat Occurrence*	CSA_X_ENTRY_ATTR_REPEAT_OCCURRENCE_NUM	2-5	1-4	**
Repeat Times*	CSA_X_ENTRY_ATTR_REPEAT_TIMES	2-5	1-4	**
Repeat Type*	CSA_X_DT_ENTRY_ATTR_REPEAT_TYPE	2-5	1-4	**
Sequence End Date*	CSA_X_ENTRY_ATTR_SEQUENCE_END_DATE	2-5	1-4	**
Showtimes*	CSA_X_ENTRY_ATTR_SHOWTIME	2-5	1-4	×
Sponsor	CSA_ENTRY_ATTR_SPONSOR	5	4	×
Start Date	CSA_ENTRY_ATTR_START_DATE	2-5	1-4	×
Status	CSA_ENTRY_ATTR_STATUS	2-5	1-4	×
Subtype	CSA_ENTRY_ATTR_SUBTYPE	2-5	1-4	×
Summary	CSA_ENTRY_ATTR_SUMMARY	2-5	1-4	×
Transparency	CSA_ENTRY_ATTR_TIME_TRANSPARENCY	5	4	×
Type	CSA_ENTRY_ATTR_TYPE	2-5	1-4	○***

* CDE のみ

** データ・バージョン 1 から 3 については、この属性は指定または変更できます。ただし、データ・バージョン 4 については読み取り専用です。データ・バージョン 4 では、エントリ属性 CSA_ENTRY_ATTR_RECURRENCE_RULE から値が取られます。

***カレンダ作成時に指定し、その後は読み取り専用になります。

次のカレンダ属性はサポートされません。

```
CSA_ENTRY_ATTR_ATTENDEE_LIST
CSA_ENTRY_ATTR_EXCEPTION_RULE
CSA_ENTRY_ATTR_RECURRING_DATES
CSA_ENTRY_ATTR_SEQUENCE_NUMBER
```

次の節では、表 10–5 にリストした項目属性について、追加の情報を提供します。

- Organizer

  項目の主催者は、csa_add_entry() を呼び出してカレンダに項目を追加するアプリケーションを実行しているユーザに設定されます。読み取り専用で、項目の追加後に変更できません。

- Reference Identifier

  項目の参照識別子は、カレンダ内の項目の固有な識別子と、カレンダの名前と位置を含んだ文字列です。形式は n:calendar@location です。n は、カレンダ内の項目を固有に識別する番号です。calendar は、カレンダ名です。location は、カレンダが格納されているマシン名です。

- Status

  CDE では、次の追加のステータス値を定義します。

```
CSA_X_DT_STATUS_ACTIVE
CSA_X_DT_STATUS_DELETE_PENDING
CSA_X_DT_STATUS_ADD_PENDING
CSA_X_DT_STATUS_COMMITTED
CSA_X_DT_STATUS_CANCELLED
```

- Type

  この値は読み取り専用で、項目の追加後に変更できません。CDE では、次の追加の型の値を定義します。

```
CSA_X_DT_TYPE_OTHER
```

CDE エントリ属性

CDE 定義済み項目属性は、次のとおりです。

- Show Time

  この属性の値は、項目の開始時間と終了時間をユーザに対して表示するかどうかを示します。csa_update_entry_attributes() により変更できます。この属性は、CSA_VALUE_SINT32 型の属性です。

- Repeat Type

  項目の反復の頻度、すなわち、どれくらいの間隔で項目を繰り返すかを示します。

  この属性は、CSA_VALUE_UINT32 型の属性です。

  次の値が定義されています。

```
CSA_X_DT_REPEAT_ONETIME
CSA_X_DT_REPEAT_DAILY
CSA_X_DT_REPEAT_WEEKLY
CSA_X_DT_REPEAT_BIWEEKLY
CSA_X_DT_REPEAT_MONTHLY_BY_WEEKDAY
CSA_X_DT_REPEAT_MONTHLY_BY_DATE
CSA_X_DT_REPEAT_YEARLY
CSA_X_DT_REPEAT_EVERY_NDAY
CSA_X_DT_REPEAT_EVERY_NWEEK
CSA_X_DT_REPEAT_EVERY_NMONTH
CSA_X_DT_REPEAT_MON_TO_FRI
CSA_X_DT_REPEAT_MONWEDFRI
CSA_X_DT_REPEAT_TUETHUR
CSA_X_DT_REPEAT_WEEKDAYCOMBO
CSA_X_DT_REPEAT_OTHER
CSA_X_DT_REPEAT_OTHER_WEEKLY
CSA_X_DT_REPEAT_OTHER_MONTHLY
CSA_X_DT_REPEAT_OTHER_YEARLY
```

- Repeat Times

  この属性は、項目を繰り返す回数を示します。この属性は、CSA_VALUE_UINT32 型の属性です。

- Repeat Interval

この属性は、Repeat Type の CSA_X_DT_REPEAT_EVERY_NDAY、CSA_X_DT_REPEAT_EVERY_NWEEK、または CSA_X_DT_REPEAT_EVERY_NMONTH の何倍で項目を繰り返すかを示します。たとえば、この属性の値が 3 であり、Repeat Type が CSA_X_DT_REPEAT_EVERY_NWEEK の場合には、項目は 3 週間ごとに繰り返されます。この属性は、CSA_VALUE_UINT32 型の属性です。

- Repeat Occurrence Number

  項目の Repeat Type が CSA_X_DT_REPEAT_MONTHLY_BY_WEEKDAY の場合、この属性は、項目を繰り返す週を示します。この属性は、CSA_VALUE_SINT32 型の属性です。

- Sequence End Date

  この項目属性は、シーケンスの終了日付を示します。この属性は、CSA_VALUE_DATE_TIME 型の属性です。

反復情報のエントリ属性

データ・バージョン 1 から 3 については、次の属性を使用してエントリの反復情報を指定します。すべて読み取りおよび書き込み属性です。

```
CSA_X_DT_ENTRY_ATTR_REPEAT_TYPE
CSA_X_DT_ENTRY_ATTR_REPEAT_TIMES
CSA_X_DT_ENTRY_ATTR_REPEAT_INTERVAL
CSA_X_DT_ENTRY_ATTR_REPEAT_OCCURRENCE_NUM
CSA_X_DT_ENTRY_ATTR_SEQUENCE_END_DATE
```

データ・バージョン 4 については、エントリ属性 CSA_ENTRY_ATTR_RECURRENCE_RULE と CSA_ENTRY_ATTR_EXCEPTION_DATES を使用してカレンダ・エントリの反復情報を指定します。CSA_ENTRY_ATTR_RECURRENCE_RULE 属性の情報は、次の属性を使用して問い合わせることができます。

```
CSA_X_DT_ENTRY_ATTR_REPEAT_TYPE
CSA_X_DT_ENTRY_ATTR_REPEAT_TIMES
CSA_X_DT_ENTRY_ATTR_REPEAT_INTERVAL
CSA_X_DT_ENTRY_ATTR_REPEAT_OCCURRENCE_NUM
CSA_X_DT_ENTRY_ATTR_SEQUENCE_END_DATE
```

これらの計算された属性は、データ・バージョン 4 に対して読み取り専用です。

データ・バージョンによりサポートされる値

- CSA_ENTRY_ATTR_STATUS

  データ・バージョン 1 はこの属性をサポートしません。

  データ・バージョン 2 と 3 は次の値をサポートします。

```
CSA_X_DT_STATUS_ACTIVE
CSA_X_DT_STATUS_DELETE_PENDING
CSA_X_DT_STATUS_ADD_PENDING
CSA_X_DT_STATUS_COMMITTED
CSA_X_DT_STATUS_CANCELLED
```

  データ・バージョン 4 はすべての状態値をサポートします。

```
CSA_STATUS_ACCEPTED
CSA_STATUS_NEEDS_ACTION
CSA_STATUS_SENT
CSA_STATUS_TENTATIVE
CSA_STATUS_CONFIRMED
CSA_STATUS_REJECTED
CSA_STATUS_COMPLETED
CSA_STATUS_DELEGATED
CSA_X_DT_STATUS_ACTIVE
CSA_X_DT_STATUS_DELETE_PENDING
CSA_X_DT_STATUS_ADD_PENDING
CSA_X_DT_STATUS_COMMITTED
CSA_X_DT_STATUS_CANCELLED
```

- CSA_ENTRY_ATTR_SUBTYPE

  データ・バージョン 1 から 3 は次の値をサポートします。

```
CSA_SUBTYPE_APPOINTMENT
CSA_SUBTYPE_HOLIDAY
```

データ・バージョン 4 はすべての定義値と、アプリケーションで定義する次のような値をサポートします。

```
CSA_SUBTYPE_APPOINTMENT
CSA_SUBTYPE_CLASS
CSA_SUBTYPE_HOLIDAY
CSA_SUBTYPE_MEETING
CSA_SUBTYPE_MISCELLANEOUS
CSA_SUBTYPE_PHONE_CALL
CSA_SUBTYPE_SICK_DAY
CSA_SUBTYPE_SPECIAL_OCCASION
CSA_SUBTYPE_TRAVEL
CSA_SUBTYPE_VACATION
```

- CSA_ENTRY_ATTR_TYPE

  データ・バージョン 1 から 3 は次の値をサポートします。

```
CSA_TYPE_EVENT
CSA_TYPE_TODO
CSA_X_DT_TYPE_OTHER
```

  データ・バージョン 4 は次のすべての定義値をサポートします。

```
CSA_TYPE_EVENT
CSA_TYPE_TODO
CSA_TYPE_MEMO
CSA_X_DT_TYPE_OTHER
```

注 - 次に示すように、タイプとサブタイプの組み合わせによっては、データ・バージョン 1 から 3 でサポートします。

データ・バージョン 1 でサポートする組み合わせ

サブタイプ CSA_SUBTYPE_APPOINTMENT を持つ CSA_TYPE_EVENT
サブタイプ値を持たない CSA_X_DT_TYPE_OTHER

データ・バージョン 2 と 3 でサポートする組み合わせ

```
サブタイプ CSA_SUBTYPE_APPOINTMENT を持つ CSA_TYPE_EVENT
サブタイプ CSA_SUBTYPE_HOLIDAY を持つ CSA_TYPE_EVENT
サブタイプ値を持たない CSA_TYPE_TODO
サブタイプ値を持たない CSA_X_DT_TYPE_OTHER
```

- CSA_X_ENTRY_ATTR_REPEAT_TYPE

  データ・バージョン 1 から 3 については、この属性を使用してエントリの反復のタイプを指定します。

  データ・バージョン 1 と 2 でサポートする値

```
CSA_X_DT_REPEAT_ONETIME
CSA_X_DT_REPEAT_DAILY
CSA_X_DT_REPEAT_WEEKLY
CSA_X_DT_REPEAT_BIWEEKLY
CSA_X_DT_REPEAT_MONTHLY_BY_DATE  C
SA_X_DT_REPEAT_YEARLY
```

  データ・バージョン 3 でサポートする値

```
CSA_X_DT_REPEAT_ONETIME
CSA_X_DT_REPEAT_DAILY
CSA_X_DT_REPEAT_WEEKLY
CSA_X_DT_REPEAT_BIWEEKLY
CSA_X_DT_REPEAT_MONTHLY_BY_WEEKDAY
CSA_X_DT_REPEAT_MONTHLY_BY_DATE
CSA_X_DT_REPEAT_YEARLY
CSA_X_DT_REPEAT_EVERY_NDAY
CSA_X_DT_REPEAT_EVERY_NWEEK
CSA_X_DT_REPEAT_EVERY_NMONTH
CSA_X_DT_REPEAT_MON_TO_FRI
CSA_X_DT_REPEAT_MONWEDFRI
CSA_X_DT_REPEAT_TUETHUR
CSA_X_DT_REPEAT_WEEKDAYCOMBO  C
SA_X_DT_REPEAT_OTHER
```

  データ・バージョン 4 については、これは読み取り専用属性です。この値は、エントリ属性 CSA_ENTRY_ATTR_RECURRENCE_RULE から取られます。

関数についての一般的な情報

次の一般的な情報は、すべての関数に適用されます。

- 文字セットの制限

  カレンダ属性 CSA_CAL_ATTR_CHARACTER_SET は、カレンダのロケール情報を格納するために使用されます。

注 - ライブラリの中で渡されるテキストでの説明以外のデータは、すべて ASCII 形式でなければなりませんが、ライブラリはシングルバイトだけでなくマルチバイト文字列もサポートします。

- 属性値の型チェックは、事前定義済み属性に対してだけ行われます。
- 項目属性 CSA_ENTRY_ATTR_RECURRENCE_RULE と CSA_ENTRY_ATTR_EXCEPTION_DATES は、カレンダ項目の反復情報を指定するために使用されます。CSA_ENTRY_ATTR_RECURRENCE_RULE 属性の情報は、次の属性を使用して問い合わせることができます。

```
CSA_X_DT_ENTRY_ATTR_REPEAT_TYPE
CSA_X_DT_ENTRY_ATTR_REPEAT_TIMES
CSA_X_DT_ENTRY_ATTR_REPEAT_INTERVAL
CSA_X_DT_ENTRY_ATTR_REPEAT_OCCURRENCE_NUM
CSA_X_DT_ENTRY_ATTR_SEQUENCE_END_DATE
```

  これらの計算された属性は、読み取り専用です。

- CSA_calendar_user データ構造体は、ユーザまたはカレンダを指定します。アクセス・リスト内のユーザを指定するときには、user_name フィールドだけが使われ、他のフィールドはすべて無視されます。ログインするカレンダを指定するときには、calendar_address フィールドだけが使われ、他のフィールドはすべて無視されます。形式は calendar@location です。calendar はカレンダ名で、location はカレンダが格納されているマシン名です。
- 値の型が CSA_VALUE_ATTENDEE_LIST の属性はサポートされません。それらの値が指定された場合には CSA_E_INVALID_ATTRIBUTE_VALUE が返されます。
- CSA_reminder データ構造体の中の repeat_count フィールドと snooze_time フィールドはカレンダに格納されますが、カレンダ・サービスは

それらの値を解釈せず、関連する通知方法はサーバによって一度しか返されません。

- ユーザ・インタフェース拡張 CSA_X_UI_ID_EXT はサポートされていません。

サポートされる関数の拡張

- Xt アプリケーション・コンテキスト (CSA_X_XT_APP_CONTEXT_EXT)

  - Xt アプリケーション・コンテキストを指定する

  - csa_register_callback() により使用される

  - 入力

  item_data: Xt アプリケーション・コンテキスト (XtAppContext)

  - 出力なし

- ユーザアクセス権の取得 (CSA_X_DT_GET_USER_ACCESS_EXT)

  - カレンダに関するユーザのアクセス権を取得する

  - csa_logon() により使用される

  - 入力なし

  - 出力

  item_data: ユーザのアクセス権 (CSA_flags)

- カレンダの文字セット属性の取得 (CSA_X_DT_GET_CAL_CHARSET_EXT)

  - カレンダの文字セット属性を取得する

  - csa_logon() により使用される

  - 入力なし

  - 出力

  item_data: item_reference の文字列の長さ (CSA_uint32)

  item_reference: 文字セット (CSA_string)

- カレンダのサーバ・バージョンの取得 (CSA_X_DT_GET_SERVER_VERSION_EXT)

  - カレンダのサーバ・バージョンを取得する

  - csa_logon() と csa_list_calendars() により使用される

  - 入力なし

- 出力

item_data: サーバ・バージョン (CSA_uint32)

- カレンダのデータ・バージョンの取得 (CSA_X_DT_GET_DATA_VERSION_EXT)

- カレンダのデータ・バージョンを取得する

- csa_logon() により使用される

- 入力なし

- 出力

item_data: データ・バージョン (CSA_uint32)

管理関数

この節では、CDE でサポートされる管理関数について説明します。関数のプロトタイプと戻りコードのリストは、各関数に含まれています。詳細は、関連するマニュアル・ページを参照してください。

- 解放 - カレンダ・サービスによって割り当てられたメモリを解放します。

  プロトタイプ

```
CSA_return_code
   csa_free(
   CSA_buffer        memory
   );
```

  csa_free の戻り値

```
   CSA_SUCCESS
   CSA_E_INVALID_MEMORY
```

- カレンダのリスト - カレンダ・サーバによってサポートされるカレンダをリストします。

  プロトタイプ

```
CSA_return_code
   csa_list_calendars(
   CSA_service_reference
   calendar_service,
```

```
CSA_uint32
*number_names,
CSA_calendar_user
**calendar_names,  CSA_extension
*list_calendars_extensions);
```

サーバが実行されているホスト名が calendar_server に渡されなければなりません。

csa_list_calendars の戻り値

```
CSA_SUCCESS
CSA_E_INVALID_PARAMETER
CSA_E_UNSUPPORTED_FUNCTION_EXT
CSA_E_INSUFFICIENT_MEMORY
CSA_X_DT_E_INVALID_SERVER_LOCATION
CSA_X_DT_E_SERVICE_NOT_REGISTERED
CSA_X_DT_E_SERVER_TIMEOUT
CSA_E_FAILURE
CSA_E_NOT_SUPPORTED
CSA_E_SERVICE_UNAVAILABLE
```

- ログイン - カレンダ・サービスにログインして、カレンダとのセッションを確立します。

  プロトタイプ

```
CSA_return_code

 csa_logon(
  CSA_service_reference calendar_service,
  CSA_calendar_user      *user,
  CSA_string       password,
  CSA_string       character_set,
  CSA_string       required_csa_version,
  CSA_session_handle      *session,
  CSA_extension       *logon_extensions);
```

引き数 calendar_service、password、character_set、および required_csa_version は使用されません。

user によって指示される CSA_calendar_user 構造体の calendar_address フィールドは、ログインするカレンダを指定します。形式は calendar@location です。calendar はカレンダ名であり、location はカレンダが格納されているホスト名です。

csa_logon の戻り値

```
CSA_SUCCESS
CSA_E_INVALID_PARAMETER
CSA_E_UNSUPPORTED_FUNCTION_EXT
CSA_E_CALENDAR_NOT_EXIST
CSA_E_INSUFFICIENT_MEMORY
CSA_E_NO_AUTHORITY
CSA_X_DT_E_INVALID_SERVER_LOCATION
CSA_X_DT_E_SERVICE_NOT_REGISTERED
CSA_X_DT_E_SERVER_TIMEOUT
CSA_E_FAILURE
CSA_E_SERVICE_UNAVAILABLE
CSA_X_DT_E_BACKING_STORE_PROBLEM
```

- ログアウト - カレンダとのセッションを終了します。

  プロトタイプ

```
CSA_return_code
 csa_logoff(
  CSA_session_handle       session,
  CSA-extension        *logoff_extensions);
```

  csa_logoff の戻り値

```
CSA_SUCCESS
CSA_E_INVALID_PARAMETER
CSA_E_UNSUPPORTED_FUNCTION_EXT
CSA_E_INSUFFICIENT_MEMORY
CSA_E_INVALID_SESSION_HANDLE
```

- 構成の問い合わせ - インストールされた CSA 構成に関する情報を判断します。

  プロトタイプ

```
CSA_return_code
 csa_query_configuration(
  CSA_session_handle      session,
  CSA_enum       item,
  CSA_buffer       *reference,
  CSA_extension       *query_configuration_extensions);
```

  CDE では、次の項目はサポートされません。

```
CSA_CONFIG_CHARACTER_SET
CSA_CONFIG_LINE_TERM
CSA_CONFIG_VER_IMPLEM
```

csa_query_configuration の戻り値

```
CSA_SUCCESS
CSA_E_INVALID_ENUM
CSA_E_INVALID_PARAMETER
CSA_E_INSUFFICIENT_MEMORY
CSA_E_INVALID_SESSION_HANDLE
CSA_E_UNSUPPORTED_ENUM
CSA_E_UNSUPPORTED_FUNCTION_EXT
```

カレンダ管理関数

この節では、CDE でサポートされるカレンダ管理関数について説明します。関数のプロトタイプと戻りコードのリストは、各関数に含まれています。詳細は、関連するマニュアル・ページを参照してください。

- カレンダの追加 - カレンダ・サービスにカレンダを追加します。

  プロトタイプ

```
CSA_return_code

 csa_add_calendar(
  CSA_session_handle      session,
  CSA_calendar_user      *user,
  CSA_uint32       number_attributes,
  CSA_attribute       *calendar_attributes,
  CSA_extension       *add_calendar_extensions);
```

  最初の引き数 session は無視されます。

  user によって示される CSA_calendar_user 構造体の calendar_address フィールドは、作成されるカレンダの名前と位置を指定します。形式は calendar@location です。calendar はカレンダ名であり、location はカレンダが格納されるホスト名です (たとえば my_calendar@my_host のようになります)。

csa_add_calendar の戻り値

```
CSA_SUCCESS
CSA_E_INVALID_PARAMETER
CSA_E_UNSUPPORTED_FUNCTION_EXT
CSA_E_INSUFFICIENT_MEMORY
CSA_E_CALENDAR_EXISTS
CSA_E_NO_AUTHORITY
CSA_E_READONLY
CSA_E_INVALID_ATTRIBUTE
CSA_E_INVALID_ATTRIBUTE_VALUE
CSA_E_UNSUPPORTED_ATTRIBUTE
CSA_E_INVALID_DATE_TIME
CSA_E_DISK_FULL
CSA_X_DT_E_BACKING_STORE_PROBLEM
CSA_X_DT_E_INVALID_SERVER_LOCATION
CSA_X_DT_E_SERVICE_NOT_REGISTERED
CSA_X_DT_E_SERVER_TIMEOUT
CSA_E_FAILURE
CSA_E_SERVICE_UNAVAILABLE
```

- コールバックの呼び出し - 指定されたコールバック・リストに関連するコールバック関数を強制的に呼び出します。

  プロトタイプ

```
CSA_return_code
 csa_call_callbacks(
   CSA_session_handle      session,
   CSA_flags       reason,
   CSA_extension      *call_callbacks_extensions);
```

  csa_call_callbacks の戻り値

```
CSA_SUCCESS
CSA_E_INVALID_PARAMETER
CSA_E_UNSUPPORTED_FUNCTION_EXT
CSA_E_INVALID_FLAG
CSA_E_INSUFFICIENT_MEMORY
CSA_E_INVALID_SESSION_HANDLE
CSA_X_DT_E_MT_UNSAFE
```

- カレンダの削除 - カレンダ・サービスからカレンダを削除します。

  プロトタイプ

```
CSA_return_code
 csa_delete_calendar(
```

```
  CSA_session_handle      session,
  csa_extension       *delete_calendar_extensions);
```

csa_delete_calendar の戻り値

```
CSA_SUCCESS
CSA_E_INVALID_PARAMETER
CSA_E_UNSUPPORTED_FUNCTION_EXT
CSA_E_INSUFFICIENT_MEMORY
CSA_E_INVALID_SESSION_HANDLE
CSA_E_NOT_SUPPORTED
CSA_E_NO_AUTHORITY
CSA_X_DT_E_BACKING_STORE_PROBLEM
CSA_X_DT_E_INVALID_SERVER_LOCATION
CSA_X_DT_E_SERVICE_NOT_REGISTERED
CSA_X_DT_E_SERVER_TIMEOUT
CSA_E_FAILURE
CSA_E_SERVICE_UNAVAILABLE
```

- カレンダ属性のリスト - カレンダに関連するカレンダ属性名をリストします。

プロトタイプ

```
CSA_return_code
 csa_list_calendar_attributes(
   CSA_session_handle      session,
   CSA_uint32       *number_names,
   CSA_attribute_reference     **calendar_attributes_names,
   CSA_extension      *list_calendar_attributes_extensions);
```

csa_list_calendar_attributes の戻り値

```
CSA_SUCCESS
CSA_E_INVALID_PARAMETER
CSA_E_UNSUPPORTED_FUNCTION_EXT
CSA_E_INSUFFICIENT_MEMORY
CSA_E_INVALID_SESSION_HANDLE
CSA_E_NOT_SUPPORTED
CSA_E_NO_AUTHORITY
CSA_X_DT_E_BACKING_STORE_PROBLEM
CSA_X_DT_E_INVALID_SERVER_LOCATION
CSA_X_DT_E_SERVICE_NOT_REGISTERED
CSA_X_DT_E_SERVER_TIMEOUT
CSA_E_FAILURE
CSA_E_SERVICE_UNAVAILABLE
```

- カレンダ属性の読み取り - カレンダのカレンダ属性値を読み取り、返します。

プロトタイプ

```
CSA_return_code
 csa_read_calendar_attributes(
   CSA_session_handle      session,
   CSA_uint32       number_names,
   CSA_attribute_reference     *attribute_names,
   CSA_uint32       *number_attributes,
   CSA_attribute      **calendar_attributes,
   CSA_extension      *read_calendar_attributes_extensions);
```

csa_read_calendar_attributes の戻り値

```
CSA_SUCCESS
CSA_E_INVALID_PARAMETER
CSA_E_UNSUPPORTED_FUNCTION_EXT
CSA_E_INSUFFICIENT_MEMORY
CSA_E_INVALID_SESSION_HANDLE
CSA_X_DT_E_INVALID_SERVER_LOCATION
CSA_X_DT_E_SERVICE_NOT_REGISTERED
CSA_X_DT_E_SERVER_TIMEOUT
CSA_E_FAILURE
CSA_E_SERVICE_UNAVAILABLE
```

- コールバック関数の登録 - カレンダの中で指定された種類の更新が行われるときに実行されるコールバック関数を登録します。

  プロトタイプ

```
CSA_return_code
 csa_register_callback(
   CSA_session_handle      session,
   CSA_flags       reason,
   CSA_callback       callback,
   CSA_buffer       client_data,
   CSA_extension      *register_callback_extensions);
```

  csa_register_callbacks の戻り値

```
CSA_SUCCESS
CSA_E_INVALID_PARAMETER
CSA_E_UNSUPPORTED_FUNCTION_EXT
CSA_E_INSUFFICIENT_MEMORY
CSA_E_INVALID_SESSION_HANDLE
CSA_E_INVALID_FLAG
CSA_X_DT_E_INVALID_SERVER_LOCATION
CSA_X_DT_E_SERVICE_NOT_REGISTERED
CSA_X_DT_E_SERVER_TIMEOUT
```

(続く)

```
CSA_E_FAILURE
CSA_E_SERVICE_UNAVAILABLE
```

- コールバック関数の登録解除 - 指定されたコールバック関数の登録を解除します。

  プロトタイプ

```
CSA_return_code
 csa_unregister_callback(
  CSA_session_handle      session,
  CSA_flags       reason,
  CSA_callback       callback,
  CSA_buffer       client_data,
  CSA_extension       *unregister_callback_extensions);
```

  csa_unregister_callback の戻り値

```
CSA_SUCCESS
CSA_E_INVALID_PARAMETER
CSA_E_UNSUPPORTED_FUNCTION_EXT
CSA_E_INVALID_SESSION_HANDLE
CSA_E_INVALID_FLAG
CSA_E_CALLBACK_NOT_REGISTERED
CSA_E_FAILURE
```

- カレンダ属性の更新 - カレンダのカレンダ属性値を更新します。

  プロトタイプ

```
CSA_return_code csa_update_calendar_attributes(
 CSA_session_handle      session,
 CSA_uint32       number_attributes,
 CSA_attribute       *calendar_attributes,
 CSA_extension       *update_calendar_attributes_extensions);
```

  csa_update_calendar_attributes の戻り値

```
CSA_SUCCESS
CSA_E_INVALID_PARAMETER
CSA_E_UNSUPPORTED_FUNCTION_EXT
CSA_E_INSUFFICIENT_MEMORY
CSA_E_INVALID_SESSION_HANDLE
CSA_E_NO_AUTHORITY
CSA_E_INVALID_ATTRIBUTE_VALUE
CSA_E_INVALID_ATTRIBUTE
CSA_E_UNSUPPORTED_ATTRIBUTE
CSA_E_READONLY
CSA_E_INVALID_DATE_TIME
CSA_E_DISK_FULL
CSA_X_E_BACKING_STORE_PROBLEM
CSA_X_DT_E_INVALID_SERVER_LOCATION
CSA_X_DT_E_SERVICE_NOT_REGISTERED
CSA_X_DT_E_SERVER_TIMEOUT
CSA_E_FAILURE
CSA_E_SERVICE_UNAVAILABLE
```

項目管理関数

この節では、CDE でサポートされる項目管理関数について説明します。関数のプロトタイプと戻りコードのリストは、各関数に含まれています。詳細は、関連するマニュアル・ページを参照してください。

- 項目の追加 - 指定されたカレンダに項目を追加します。

  プロトタイプ

```
CSA_return_code
 csa_add_entry(
   CSA_session_handle      session,
   CSA_uint32       number_attributes,
   CSA_attribute      *entry_attributes,
   CSA_entry_handle      *entry,
   CSA_extension      *add_entry_extensions);
```

  csa_add_entry の戻り値

```
CSA_SUCCESS
CSA_E_INVALID_PARAMETER
CSA_E_UNSUPPORTED_FUNCTION_EXT
CSA_E_INSUFFICIENT_MEMORY
CSA_E_INVALID_SESSION_HANDLE
CSA_E_NO_AUTHORITY
CSA_E_READONLY
```

```
CSA_E_UNSUPPORTED_ATTRIBUTE
CSA_E_INVALID_ATTRIBUTE
CSA_E_INVALID_ATTRIBUTE_VALUE
CSA_E_INVALID_DATE_TIME
CSA_E_INVALID_RULE
CSA_E_DISK_FULL
CSA_X_E_BACKING_STORE_PROBLEM
CSA_X_DT_E_INVALID_SERVER_LOCATION
CSA_X_DT_E_SERVICE_NOT_REGISTERED
CSA_X_DT_E_SERVER_TIMEOUT
CSA_E_FAILURE
CSA_E_SERVICE_UNAVAILABLE
```

エントリを追加するときに指定する最小限の属性セットは次のとおりです。

データ・バージョン 1 から 3 の場合

指定する必要があるもの

```
CSA_ENTRY_ATTR_START_DATE
CSA_ENTRY_ATTR_TYPE
```

指定しないとデフォルトで設定されるもの

```
 CSA_ENTRY_ATTR_CLASSIFICATION (CSA_CLASS_PUBLIC)
CSA_ENTRY_ATTR_STATUS (CSA_X_DT_STATUS_ACTIVE)
CSA_ENTRY_ATTR_SUBTYPE (CSA_SUBTYPE_APPOINTMENT for type
      CSA_TYPE_EVENT;
this attribute is not supported for type
      CSA_TYPE_TODO)
CSA_ENTRY_ATTR_SUMMARY (NULL string)
CSA_X_ENTRY_ATTR_REPEAT_TYPE (CSA_X_REPEAT_ONETIME)
CSA_X_ENTRY_ATTR_SHOWTIME (1 => true)
```

データ・バージョン 4 の場合

指定する必要があるもの

```
CSA_ENTRY_ATTR_START_DATE
CSA_ENTRY_ATTR_TYPE
```

指定しないとデフォルトで設定されるもの

```
CSA_ENTRY_ATTR_CLASSIFICATION
(CSA_CLASS_PUBLIC)
  CSA_ENTRY_ATTR_STATUS (CSA_X_STATUS_ACTIVE)
CSA_ENTRY_ATTR_SUBTYPE (CSA_SUBTYPE_APPOINTMENT for type
      CSA_TYPE_EVENT)
CSA_ENTRY_ATTR_SUMMARY (NULL string)
  CSA_X_ENTRY_ATTR_SHOWTIME (1 =>true)
```

- 項目の削除 - 指定されたカレンダから項目を削除します。

  プロトタイプ

```
CSA_return_code
 csa_delete_entry(
  CSA_session_handle      session,
  CSA_entry_handle      entry,
  CSA_enum       delete_scope,
  CSA_extension       *delete_entry_extensions);
```

  csa_delete_entry の戻り値

```
CSA_SUCCESS
CSA_E_INVALID_ENUM
CSA_E_INVALID_PARAMETER
CSA_E_UNSUPPORTED_FUNCTION_EXT
CSA_E_INSUFFICIENT_MEMORY
CSA_E_INVALID_SESSION_HANDLE
CSA_E_INVALID_ENTRY_HANDLE
CSA_E_NO_AUTHORITY
CSA_X_DT_E_ENTRY_NOT_FOUND
CSA_E_DISK_FULL
CSA_X_E_BACKING_STORE_PROBLEM
CSA_X_DT_E_INVALID_SERVER_LOCATION
CSA_X_DT_E_SERVICE_NOT_REGISTERED
CSA_X_DT_E_SERVER_TIMEOUT
CSA_E_FAILURE
CSA_E_SERVICE_UNAVAILABLE
```

- 項目のリスト - 属性検索基準のすべてに一致するカレンダ項目をリストします。

  プロトタイプ

```
CSA_return_code
 csa_list_entries(
  CSA_session_handle      session,
  CSA_uint32       number_attributes,
```

```
  CSA_attribute       *entry_attributes,
  CSA_enum       *list_operators,
  CSA_uint32       *number_entries,
  CSA_entry_handle      **entries,
  CSA_extension       *list_entries_extensions);
```

list_operators で指定される演算子について、さらに詳しく説明します。

属性値の型 CSA_VALUE_REMINDER、CSA_VALUE_CALENDAR_USER、および CSA_VALUE_DATE_TIME_RANGE については、演算子 CSA_MATCH_ANY と CSA_MATCH_EQUAL_TO だけがサポートされます。

属性値の型 CSA_VALUE_STRING については、演算子 CSA_MATCH_ANY、CSA_MATCH_EQUAL_TO、CSA_MATCH_NOT_EQUAL_TO、および CSA_MATCH_CONTAIN だけがサポートされます。演算子 CSA_MATCH_CONTAIN は、CSA_VALUE_STRING 型の属性にだけ適用されます。

値の型が CSA_VALUE_OPAQUE_DATA、CSA_VALUE_ACCESS_LIST、CSA_VALUE_ATTENDEE_LIST、および CSA_VALUE_DATE_TIME_LIST の属性の照合はサポートされません。唯一の例外は、属性 CSA_ENTRY_ATTR_REFERENCE_IDENTIFIER です。演算子 CSA_MATCH_EQUAL_TO は、この属性に対してサポートされます。

csa_list_entries の戻り値

```
CSA_SUCCESS

CSA_E_INVALID_PARAMETER
CSA_E_UNSUPPORTED_FUNCTION_EXT

CSA_E_INSUFFICIENT_MEMORY
CSA_E_INVALID_SESSION_HANDLE
CSA_E_NO_AUTHORITY

CSA_E_INVALID_ATTRIBUTE_VALUE
CSA_E_INVALID_DATE_TIME
CSA_E_INVALID_ENUM

CSA_E_UNSUPPORTED_ENUM
CSA_X_E_BACKING_STORE_PROBLEM

CSA_X_DT_E_INVALID_SERVER_LOCATION
CSA_X_DT_E_SERVICE_NOT_REGISTERED

CSA_X_DT_E_SERVER_TIMEOUT
CSA_E_FAILURE
```

(続く)

```
CSA_E_SERVICE_UNAVAILABLE
```

- 項目属性のリスト - 指定された項目に関連する項目の属性名をリストします。

  プロトタイプ

```
CSA_return_code
 csa_list_entry_attributes(
  CSA_session_handle      session,
  CSA_entry_handle      entry,
  CSA_uint32       *number_names,
  CSA_attribute_reference     **entry_attribute_names,
  CSA_extension       *list_entry_attributes_extensions);
```

  csa_list_entry_attributes の戻り値

```
CSA_SUCCESS

CSA_E_INVALID_PARAMETER
CSA_E_UNSUPPORTED_FUNCTION_EXT

CSA_E_INSUFFICIENT_MEMORY
CSA_E_INVALID_SESSION_HANDLE

CSA_E_INVALID_ENTRY_HANDLE
CSA_X_E_ENTRY_NOT_FOUND

CSA_X_E_BACKING_STORE_PROBLEM
CSA_X_DT_E_INVALID_SERVER_LOCATION

CSA_X_DT_E_SERVICE_NOT_REGISTERED
CSA_X_DT_E_SERVER_TIMEOUT
CSA_E_FAILURE

CSA_E_SERVICE_UNAVAILABLE
```

- 項目シーケンスのリスト - カレンダ項目に関連する再帰的カレンダ項目をリストします。

  プロトタイプ

```
CSA_return_code
 csa_list_entry_sequence(
   CSA_session_handle      session,
   CSA_entry_handle      entry,
   CSA_date_time_range      time_range,
```

```
  CSA_uint32       *number_entries,
  CSA_entry_handle      **entry_list,
  CSA_extension      *list_entry_sequences_extensions);
```

指定された項目が一回限りの項目の場合には、CSA_E_INVALID_PARAMETER が返されます。

csa_list_entry_sequence の戻り値

```
CSA_SUCCESS

CSA_E_INVALID_PARAMETER
CSA_E_UNSUPPORTED_FUNCTION_EXT

CSA_E_INSUFFICIENT_MEMORY
CSA_E_INVALID_SESSION_HANDLE

CSA_E_INVALID_ENTRY_HANDLE
CSA_E_INVALID_DATE_TIME  CSA_X_E_ENTRY_NOT_FOUND

CSA_X_E_BACKING_STORE_PROBLEM
CSA_X_DT_E_INVALID_SERVER_LOCATION

CSA_X_DT_E_SERVICE_NOT_REGISTERED
CSA_X_DT_E_SERVER_TIMEOUT  CSA_E_FAILURE

CSA_E_SERVICE_UNAVAILABLE
```

- 項目属性の読み取り - 指定された項目のカレンダ項目属性値を読み取り、返します。

  プロトタイプ

```
CSA_return_code
 csa_read_entry_attributes(
   CSA_session_handle      session,
   CSA_entry_handle      entry,
   CSA_uint32       number_names,
   CSA_attribute_reference     *attribute_names,
   CSA_uint32       *number_attributes,
   CSA_attribute      **entry_attributes,
   CSA_extension      *read_entry_attributes_extensions);
```

  csa_read_entry_attributes の戻り値

```
CSA_SUCCESS

CSA_E_INVALID_PARAMETER
CSA_E_UNSUPPORTED_FUNCTION_EXT

CSA_E_INSUFFICIENT_MEMORY
CSA_E_INVALID_SESSION_HANDLE

CSA_E_INVALID_ENTRY_HANDLE
CSA_X_E_ENTRY_NOT_FOUND

CSA_X_E_BACKING_STORE_PROBLEM
CSA_X_DT_E_INVALID_SERVER_LOCATION

CSA_X_DT_E_SERVICE_NOT_REGISTERED
CSA_X_DT_E_SERVER_TIMEOUT  CSA_E_FAILURE

CSA_E_SERVICE_UNAVAILABLE
```

- 次の通知方法の読み取り - 特定の時間を基準として、指定されたカレンダの中の特定の種類の次の通知方法を読み取ります。

  プロトタイプ

```
CSA_return_code
 csa_read_next_reminder(
  CSA_session_handle      session,
  CSA_uint32       number_names,
  CSA_attribute_reference     *reminder_names,
  CSA_date_time       given_time,
  CSA_uint32       *number_reminders,
  CSA_reminder_reference     **reminder_references,
  CSA_extension       *read_next_reminder_extensions);
```

  csa_read_next_reminder の戻り値

```
CSA_SUCCESS

CSA_E_INVALID_PARAMETER
CSA_E_UNSUPPORTED_FUNCTION_EXT

CSA_E_INSUFFICIENT_MEMORY
CSA_E_INVALID_SESSION_HANDLE

CSA_E_INVALID_DATE_TIME
CSA_E_NO_AUTHORITY  CSA_X_E_BACKING_STORE_PROBLEM

CSA_X_DT_E_INVALID_SERVER_LOCATION
CSA_X_DT_E_SERVICE_NOT_REGISTERED
```

(続く)

```
CSA_X_DT_E_SERVER_TIMEOUT
CSA_E_FAILURE

CSA_E_SERVICE_UNAVAILABLE
```

- 項目属性の更新 - カレンダ項目属性を更新します。

  プロトタイプ

```
CSA_return_code
 csa_update_entry_attributes(
  CSA_session_handle      session,
  CSA_entry_handle      entry,
  CSA_enum       update_scope,
  CSA_boolean       update_propagation,
  CSA_uint32       number_attributes,
  CSA_attribute       *entry_attributes,
  CSA_entry_handle      *new_entry,
  CSA_extension       *update_entry_attributes_extensions);
```

  更新の伝達はサポートされません。update_propagation 引き数は CSA_FALSEに設定してください。

  csa_update_entry_attributes の戻り値

```
CSA_SUCCESS

CSA_E_INVALID_PARAMETER
CSA_E_UNSUPPORTED_FUNCTION_EXT

CSA_E_INSUFFICIENT_MEMORY
CSA_E_INVALID_SESSION_HANDLE

CSA_E_INVALID_ENTRY_HANDLE
CSA_E_NO_AUTHORITY
CSA_E_READONLY

CSA_E_INVALID_ENUM
CSA_E_UNSUPPORTED_ATTRIBUTE
CSA_E_INVALID_ATTRIBUTE

CSA_E_INVALID_ATTRIBUTE_VALUE
CSA_E_INVALID_DATE_TIME
CSA_E_INVALID_RULE

CSA_E_DISK_FULL
```

(続く)

```
CSA_X_E_BACKING_STORE_PROBLEM

CSA_X_DT_E_INVALID_SERVER_LOCATION
CSA_X_DT_E_SERVICE_NOT_REGISTERED

CSA_X_DT_E_SERVER_TIMEOUT
CSA_E_FAILURE

CSA_E_SERVICE_UNAVAILABLE
```

- サポートされない関数

  次の関数は CDE でサポートされません。CSA_E_NOT_SUPPORTED だけが返されます。

```
csa_add_event
csa_add_memo
csa_add_todo
csa_free_time_search
csa_look_up
csa_restore
csa_save
```

コーディング例

カレンダのリストおよび出力

例 **10–1**　サーバがサポートするカレンダをリストし出力する

```
Example:
List and print out the calendars supported by a server.
    Free memory returned by a CSA function.
list_calendar()
{
 CSA_return_code  stat;
CSA_uint32  i, number;
 CSA_calendar_user *calendars;
 char   *host;

  /* ネットワーク上のいくつかのマシンを指定します */
 host = ''somehost'';

  stat= csa_list_calendars(host, &number, &calendars, NULL);
  for (i = 0; i< number; i++) {
  /* calendar_address フィールドには、カレンダのアドレスが
   * user@host の書式で入っています   */
 printf(''%d: %s\n'', i, calendars[i].calendar_address);
  }

  /* 例: CSA 関数により返されるメモリを解放する
  * csa_list_calendars により返されるメモリを解放します
  */
 stat = csa_free(calendars);
}
```

カレンダの追加

例 **10–2**　activity という名前のカレンダをホスト host1 に追加する

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

CSA_access_rights *setup_access_list() {
 CSA_access_rights *ptr1, *ptr2;
  /* 任意のユーザに、公開および非公開エントリを表示する許可を与え、
  * ユーザ user1 に、公開エントリを表示し、挿入する許可を与えます。
  * 特別なユーザ名 world は任意のユーザを意味します。
  */

ptr2 = (CSA_access_rights *)calloc(1, sizeof(CSA_access_rights));
ptr2->user = (CSA_calendar_user *)calloc(1, sizeof(CSA_calendar_user));
ptr2->user->user_name = strdup(``world'');
ptr2->user->user_type = CSA_USER_TYPE_INDIVIDUAL;
ptr2->flags = CSA_VIEW_PUBLIC_ENTRIES | CSA_VIEW_CONFIDENTIAL_ENTRIES;
ptr1 = (CSA_access_rights *)calloc(1, sizeof(CSA_access_rights));
ptr1->user = (CSA_calendar_user *)calloc(1, sizeof(CSA_calendar_user));
ptr1->user->user_name = strdup(``user1'');
ptr1->user->user_type = CSA_USER_TYPE_INDIVIDUAL;
ptr1->flags = CSA_VIEW_PUBLIC_ENTRIES | CSA_INSERT_PUBLIC_ENTRIES;
ptr1->next = ptr2;
}

void destroy_access_list(CSA_access_rights *list)
{
 CSA_access_rights *ptr;

  while (list != NULL) {
  ptr = list->next;

   if (list->user) {
   if (list->user->user_name)
   free(list->user->user_name);
   free(list->user);
  }  free(list);
   list = ptr;
 }
}
add_calendar()
{
 CSA_return_code  stat;
 CSA_calendar_user caddr;
 CSA_attribute  attr;
 CSA_attribute_value attr_val;
```

```
  /* 追加するカレンダを指定します。 */

  caddr.user_name = NULL;
  caddr.user_type = NULL;
  caddr.calendar_address = ``activity@host1'';
  caddr.calendar_user_extensions = NULL;

  /* アクセス・リストを設定します */
  attr_val.type  = CSA_VALUE_ACCESS_LIST;
  attr_val.item.access_list_value = setup_access_list();
  attr.name  = CSA_CAL_ATTR_ACCESS_LIST;
  attr.value  = &attr_val;
  attr.attribute_extensions = NULL;
   stat = csa_add_calendar(NULL, &caddr, 1, &attr, NULL);
   destroy_access_list(attr_val.item.access_list_value);
}
```

カレンダへのログイン

例 **10–3**　カレンダにログインする

```
CSA_session_handle cal;

logon()
{
 CSA_return_code  stat;
 CSA_calendar_user caddr;
 CSA_flags  access;
 CSA_extension  logon_exts[2];
 CSA_X_COM_support check_support[2];

  /* ログインするカレンダを指定します */
caddr.user_name = NULL;
caddr.user_type = CSA_USER_TYPE_INDIVIDUAL;
caddr.calendar_address = ``user@host'';
caddr.calendar_user_extensions = NULL;

  /* (CSA_X_DT_GET_USER_ACCESS_EXT) を指定して、
  * カレンダに関するユーザのアクセス権を取得します。
  */
logon_exts[0].item_code  = CSA_X_DT_GET_USER_ACCESS_EXT;
logon_exts[0].item_data  = 0;
logon_exts[0].item_reference = NULL;
logon_exts[0].extension_flags = NULL;
```

```
 /* CSA_X_COM_SUPPORT_EXT 拡張を指定して、
  * CSA_X_XT_APP_CONTEXT_EXT 拡張と、CSA_X_UI_ID_EXT 拡張が
  * サポートされているかどうかをチェックします。
  */
check_support[0].item_code = CSA_X_XT_APP_CONTEXT_EXT;
check_support[0].flags  = NULL;
check_support[1].item_code = CSA_X_UI_ID_EXT;
check_support[1].flags  = NULL;
logon_exts[1].item_code  = CSA_X_COM_SUPPORT_EXT;
logon_exts[1].item_data  = 2;
logon_exts[1].item_reference = (CSA_buffer)check_support;
logon_exts[0].extension_flags = CSA_EXT_LAST_ELEMENT;

  stat = csa_logon(NULL, &caddr, NULL, NULL, NULL, &cal, logon_exts);

  if (stat == CSA_SUCCESS) {
  access = (CSA_flags)get_access_ext.item_data;
   if (check_support[0].flag & CSA_X_COM_SUPPORTED)
   printf(``The CSA_X_XT_APP_CONTEXT_EXT extension is supported\n'');
   if (check_support[1].flag & CSA_X_COM_SUPPORTED)
   printf(``The CSA_X_UI_ID_EXT extension is supported\n'');
 }
}
```

カレンダ・セッションの終了

例 10–4　カレンダ・セッションを終了する

```
logoff()
{
 CSA_return_code  stat;

  /* セッションが必要ない場合は、csa_logoff を呼び出すことにより終了できます。
  * 前の例で csa_logon により返されたセッションを終了します。
  */
 stat = csa_logoff(cal, NULL);
}
```

カレンダの削除

例 10–5　カレンダを削除する

```
delete_calendar()
{
 /* csa_logon() を呼び出すことによりカレンダ・セッションを確立後、
  * csa_delete_calender() を使用してカレンダを削除できます。
  */
 CSA_return_code  stat;
  stat = csa_delete_calendar(cal, NULL);
}
```

カレンダ・エントリの追加

例 **10–6**　カレンダ・エントリを追加する

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

CSA_return_code  stat;
CSA_session_handle cal;
CSA_attribute  attrs[9];
CSA_attribute_value attr_val[9];
CSA_reminder  audio;
CSA_reminder  mail;
CSA_entry_handle new_entry;
int  i;

i = 0;

/* 開始日の属性。この属性にはデフォルトがないため、必ず指定してください。
 * CSA_date_time 値は、ISO 8601 規格にある UTC ベースの日付と時間を指定します。
 */
attrs[i].name = CSA_ENTRY_ATTR_START_DATE;
 attrs[i].value = &attr_val[i];
 attrs[i].attribute_extensions = NULL;
 attr_val[i].type = CSA_VALUE_DATE_TIME;
 attr_val[i].item.date_time_value = iso8601time(time(NULL));
 i++;
```

```
/* 終了日の属性。
 * 指定しないと、エントリは終了日の属性を持ちません。
 */
attrs[i].name = CSA_ENTRY_ATTR_END_DATE;
 attrs[i].value = &attr_val[i];
 attrs[i].attribute_extensions = NULL;
 attr_val[i].type = CSA_VALUE_DATE_TIME;
 attr_val[i].item.date_time_value = iso8601time(time(NULL) + 3600);
 i++;

/* 分類属性。
 * 指定しないと、デフォルト値は CSA_CLASS_PUBLIC になります。
 */
attrs[i].name = CSA_ENTRY_ATTR_CLASSIFICATION;
attrs[i].value = &attr_val[i];
attrs[i].attribute_extensions = NULL;
attr_val[i].type = CSA_VALUE_UINT32;
attr_val[i].item.sint32_value = CSA_CLASS_CONFIDENTIAL;
i++;

/* タイプ属性。
 * この属性にはデフォルトがないため、必ず指定してください。
 */
attrs[i].name = CSA_ENTRY_ATTR_TYPE;
attrs[i].value = &attr_val[i];
attrs[i].attribute_extensions = NULL;
attr_val[i].type = CSA_VALUE_UINT32;
attr_val[i].item.sint32_value = CSA_TYPE_EVENT;
i++;

/* サブタイプ属性。
 * 指定しないと、デフォルト値は CSA_SUBTYPE_APPOINTMENT になります。
 */
attrs[i].name = CSA_ENTRY_ATTR_SUBTYPE;
attrs[i].value = &attr_val[i];
attrs[i].attribute_extensions = NULL;
attr_val[i].type = CSA_VALUE_STRING;
attr_val[i].item.string_value = CSA_SUBTYPE_APPOINTMENT;
i++;

/* サマリ属性 */
attrs[i].name = CSA_ENTRY_ATTR_SUMMARY;
attrs[i].value = &attr_val[i];
attrs[i].attribute_extensions = NULL;
attr_val[i].type = CSA_VALUE_STRING;
attr_val[i].item.string_value = argv6;
attrs[i].attribute_extensions = NULL;
i++;
```

```
/* 反復規則属性。
 * 指定しないと、エントリは 1 回だけのエントリになります。反復規則 D1 #3 は、
 * エントリが毎日 3 日間繰り返されるよう指定します。
 */
attrs[i].name = CSA_ENTRY_ATTR_RECURRENCE_RULE;
attrs[i].value = &attr_val[i];
attrs[i].attribute_extensions = NULL;
attr_val[i].type = CSA_VALUE_STRING;
attr_val[i].item.string_value = argv7;
i++;

/* オーディオ通知属性。
 * 通知プログラムのリード・タイムは、ISO 8601 規格にある CSA_time_duration の値で
 * 指定します。
 * たとえば、5 分間のリード・タイムは、文字列 +PT300S と表します。
 * マイナス 5 分間のリード・タイムは、-PT300S と表します。
 */
attrs[i].name = CSA_ENTRY_ATTR_AUDIO_REMINDER;
attrs[i].value = &attr_val[i];
attrs[i].attribute_extensions = NULL;
attr_val[i].type = CSA_VALUE_REMINDER;
attr_val[i].item.reminder_value = &audio;
memset((void *)&audio, NULL, sizeof(audio));
audio.lead_time = ''+PT300S''; i++;

/* メール通知属性。
 * 電子メール・アドレスは、reminder_data フィールドに指定します。
 * この通知プログラムのリード・タイムは 1 日。
 */
attrs[i].name = CSA_ENTRY_ATTR_MAIL_REMINDER;
attrs[i].value = &attr_val[i];
attrs[i].attribute_extensions = NULL;
attr_val[i].type = CSA_VALUE_REMINDER;
attr_val[i].item.reminder_value = &mail;
memset((void *)&mail, NULL, sizeof(mail));
mail.lead_time = ''+PT86400S'';
mail.reminder_data.data = ''someuser@somehost'';
mail.reminder_data.size = strlen(mail.reminder_data.data);
i++;

/* 指定した属性値でエントリを追加します */
stat = csa_add_entry(cal, i, attrs, &newentry, NULL);
if (stat == CSA_SUCCESS)
 csa_free((CSA_buffer)newentry);
```

カレンダのエントリの検索

例 **10–7**　カレンダのエントリを検索する／属性値を読み取る

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

CSA_return_code  stat;
CSA_session_handle cal;
CSA_attribute  attrs[4];
CSA_attribute_value attr_val[4];
CSA_enum  ops[4];
CSA_uint32  i;
CSA_uint32  num_entries;
CSA_entry_handle *entries;
CSA_uint32  num_attributes;
CSA_attribute  *entry_attrs;

/* 次の判断基準ですべてのエントリを検索します。
 * UTC 時間 1996 年 8 月のすべてのアポイント
 * 開始日は、UTC 時間 1996 年 8 月 1 日 00:00:00 以降
 * 開始日は、UTC 時間 1996 年 9 月 1 日 00:00:00 より前
 * タイプは CSA_TYPE_EVENT
 * サブタイプは CSA_SUBTYPE_APPOINTMENT
 */

i = 0;

/* 開始日は、UTC 時間 1996 年 8 月 1 日 00:00:00 以降 */
attrs[i].name = CSA_ENTRY_ATTR_START_DATE;
attrs[i].value = &attr_val[i];
attrs[i].attribute_extensions = NULL;
attr_val[i].type = CSA_VALUE_DATE_TIME;
attr_val[i].item.date_time_value = ''19960801T000000Z'';
ops[i] = CSA_MATCH_GREATER_THAN_OR_EQUAL_TO;
i++;

/* 開始日は、UTC 時間 1996 年 9 月 1 日 00:00:00 より前 */
attrs[i].name = CSA_ENTRY_ATTR_START_DATE;
attrs[i].value = &attr_val[i];
attrs[i].attribute_extensions = NULL;
attr_val[i].type = CSA_VALUE_DATE_TIME;
attr_val[i].item.date_time_value = ''19960901T000000Z''
ops[i] = CSA_MATCH_LESS_THAN;
i++;

/* タイプは CSA_TYPE_EVENT */
attrs[i].name = CSA_ENTRY_ATTR_TYPE;
attrs[i].value = &attr_val[i];
attrs[i].attribute_extensions = NULL;
attr_val[i].type = CSA_VALUE_UINT32;
attr_val[i].item.sint32_value = CSA_TYPE_EVENT;
ops[i] = CSA_MATCH_EQUAL_TO;
i++;
```

```
/* サブタイプは CSA_SUBTYPE_APPOINTMENT */
attrs[i].name = CSA_ENTRY_ATTR_SUBTYPE;
attrs[i].value = &attr_val[i];
attrs[i].attribute_extensions = NULL;
attr_val[i].type = CSA_VALUE_STRING;
attr_val[i].item.string_value = CSA_SUBTYPE_APPOINTMENT;
ops[i] = CSA_MATCH_EQUAL_TO;
i++;

/* 検索の実行 */
stat = csa_list_entries(csa, i, attrs, ops, &num_entries, &entries, NULL);

 if (stat == CSA_SUCCESS) {
   for (i = 0; i < num_entries; i++) {
/* エントリのすべての属性値を取得します。
 * number_names に 0 を指定し、attribute_names に NULL を指定すると、
 * すべての属性値が返されます。
 */
  stat = csa_read_entry_attributes(cal, entries[i], 0, NULL,
     &num_attributes, &entry_attrs,
     NULL); if (stat == CSA_SUCCESS) {
   /* 返された属性値を使用し、終了時にメモリを解放します
    */
    csa_free(entry_attrs);
 } else {
   /* ハンドル・エラー */
 }
   }
} else {
   /* ハンドル・エラー */
}

Example: Change the end time of the returned appointments to be
  one hour later.

CSA_attribute_reference name = CSA_ENTRY_ATTR_END_DATE;
char  buffer[80];
time_t  endtime;
CSA_entry_handle new_entry;
```

```
for (i = 0; i < num_entries; i++) {
    /* アポイントの終了時間を取得します */
    stat = csa_read_entry_attributes(cal, entries[i], 0, &name,
       &num_attributes, &entry_attrs, NULL);
    if (stat == CSA_SUCCESS) {
 /* 終了時間を 1 時間後に変更します */
 from_iso8601_time(entry_attrs[0].value->item.date_time_value, &endtime);
 endtime += 60*60 /* 1 時間の秒数 */
 to_iso8601_time(endtime, buffer);
   attrs[0].name = CSA_ENTRY_ATTR_END_DATE;
   attrs[0].value = &attr_val[i];
   attrs[0].attribute_extensions = NULL;
   attr_val[0].type = CSA_VALUE_DATE_TIME;
 attr_val[0].item.date_time_value = buffer;
  stat = csa_update_entry_attributes(cal, entries[0], CSA_SCOPE_ALL,
        CSA_FALSE, 1, attrs, &new_entry,
       NULL);
 if (stat == CSA_SUCCESS) {
    csa_free(new_entry);
 } else {
    /* ハンドル・エラー */
 }
  csa_free(entry_attrs);
    } else {
    /* ハンドル・エラー */
    }
}
```

例 **10–8**　返されたアポイントの終了時間を 1 時間後に変更する

```
CSA_attribute_reference name = CSA_ENTRY_ATTR_END_DATE;
char  buffer[80];
time_t  endtime;
CSA_entry_handle new_entry;

for (i = 0; i < num_entries; i++) {
    /* アポイントの終了時間を取得します */
    stat = csa_read_entry_attributes(cal, entries[i], 0, &name,
        &num_attributes, &entry_attrs, NULL);
    if (stat == CSA_SUCCESS) {
 /* 終了時間を 1 時間後に変更します */
 from_iso8601_time(entry_attrs[0].value->item.date_time_value, &endtime);
 endtime += 60*60 /* 1 時間の秒数 */
 to_iso8601_time(endtime, buffer);

  attrs[0].name = CSA_ENTRY_ATTR_END_DATE;
 attrs[0].value = &attr_val[i];
 attrs[0].attribute_extensions = NULL;
 attr_val[0].type = CSA_VALUE_DATE_TIME;
 attr_val[0].item.date_time_value = buffer;
  stat = csa_update_entry_attributes(cal, entries[0], CSA_SCOPE_ALL,
      CSA_FALSE, 1, attrs, &new_entry,
      NULL);  if (stat == CSA_SUCCESS) {
    csa_free(new_entry);
 } else {
    /* ハンドル・エラー */
 }
  csa_free(entry_attrs);
    } else {
    /* ハンドル・エラー */
   }
}
```

コールバックの登録および通知プログラムの保持

例 **10–9**　コールバックを登録し、通知プログラムを保持する

```
/* このコーディング例では、csa_register_callback、csa_read_next_reminder、
 * および csa_call_callbacks の使用方法について示します。
 * 基本となるコードでは、イベント CSA_CB_ENTRY_ADDED、CSA_CB_ENTRY_DELETED、
 * および CSA_CB_ENTRY_UPDATED 用のコールバック・ルーチンを登録します。
 * また、別のコールバック・ルーチンを、CSA_CB_CALENDAR_ATTRIBUTE_UPDATE
 * イベント用に登録します。
 * 通知デリバリ用のタイマを設定する方法も示します。
 * ISO 8601 書式の時間表記を UTC 1970 年 1 月 1 日 00:00:00 からの経過秒数を示す
 * ティック数への変換用の 2 つのユーティリティ・ルーチンも含みます。
 */

#include <csa/csa.h>
#include <time.h>
#include <unistd.h>

CSA_session_handle cal; /* カレンダ・セッション */
time_t  run_time; /* 通知プログラムを実行する時間 */
CSA_uint32  num_rems; /* 返される通知数 */
CSA_reminder_reference *rems;  /* 通知情報の配列 */

void
set_up_callback_handler()
{
 CSA_return_code  stat;
 CSA_flags  flags;

 /* Xt ベースのアプリケーションは、コールバック・ルーチンが非同期に呼び出せるように
 * するため、CSA_X_XT_APP_CONTEXT_EXT 拡張を使用して、
 * Xt アプリケーションのコンテキストを指定できます。
 *
 * CSA_extension callback_ext;
 * callback_ext.item_code = CSA_X_XT_APP_CONTEXT_EXT;
 *  callback_ext.item_data = (CSA_uint32)application_context;
 * callback_ext.extension_flags = CSA_EXT_LAST_ELEMENT;
 *
 * callback_ext を最後のパラメータとして、csa_register_callback に渡す。
 */

  flags = CSA_CB_ENTRY_ADDED|CSA_CB_ENTRY_DELETED|CSA_CB_ENTRY_UPDATED;
 stat = csa_register_callback(cal, flags, entry_update_callback,
  NULL, NULL);

  if (stat != CSA_SUCCESS) {
 /* エラー処理コード */
 }
```

```
  stat = csa_register_callback(cal, CSA_CB_CALENDAR_ATTRIBUTE_UPDATED,
  calendar_update_callback, NULL, NULL);
  if (stat != CSA_SUCCESS) {
 /* エラー処理コード */
 }
}

/* * このルーチンはライブラリをポーリングし、特定のイベントが発生していた場合に、
 * 登録されているコールバックを呼び出します。
 * アプリケーションが CSA_X_XT_APP_CONTEXT_EXT 拡張を使用してコールバックの
 * 呼び出しを非同期に設定していない場合、csa_call_callbacks を呼び出して、
 * コールバックの呼び出しを強制する必要があります。
 */

check_events(CSA_flags event)
{
 csa_call_callbacks(cal, event, NULL);
}

 /*
 * これは、イベント CSA_CB_ENTRY_ADDED、CSA_CB_ENTRY_ADDED、および
 * CSA_CB_ENTRY_UPDATED のためのコールバック・ルーチンです。
 */
void
entry_update_callback(
 CSA_session_handle cal,
 CSA_flags flags,
 CSA_buffer call_data,
 CSA_buffer client_data,
 CSA_extension *ext)
{
 /* エントリが追加、削除、または更新されます。
 * このコールバック・ルーチンで行う可能性があるのは次のとおりです。
 *
 * 1. カレンダ表示の更新
 * 2. ユーザ自身のカレンダの場合は、通知情報の更新
 *
 * このルーチンのコーディング例は通知情報を更新します。
  */ reset_reminder();
}
 /*
 * これは CSA_CB_CALENDAR_ATTRIBUTE_UPDATED イベントのための
 * コールバック・ルーチンです。
 */
void
calendar_update_callback(
 CSA_session_handle cal,
 CSA_flags flags,
 CSA_buffer call_data,
 CSA_buffer client_data,
 CSA_extension *ext)
{
 /* カレンダ属性を更新する */
}
```

```
  /*
 * このルーチンは通知情報を更新します。
 *  - 既存の情報がある場合は削除する
 *  - csa_read_next_reminder() を呼び出して、次に転送する通知を取得する
 *  - 実行時間をチェックし、タイマーを設定する
 */
void
reset_reminder() {
 CSA_return_code  stat;
 time_t   current_time;
 char   isotime[BUFSIZ];
 CSA_uint32  number_reminders;
 CSA_reminder_reference *reminders;

  current_time = time(NULL);

/* 既存の情報を削除する */
 if (rems) {
 /* この比較の目的は、最後に実行された時刻から、現在の時刻までの間に実行された
  * 通知が失われていないかを確認することです。
   */
  if (current_time > run_time)
   current_time = run_time;

   csa_free((CSA_buffer)rems);
 }

  to_iso8601_time(current_time, isotime);
 stat = csa_read_next_reminder(cal, 0, NULL, isotime,
  &number_reminders, &reminders, NULL);

  if (stat == CSA_SUCCESS && num_rems > 0) {
  num_rems = number_reminders;
  rems = reminders;

 /* タイマーを設定し、通知を転送する。
  * sigset() を使用して、SIGALRM シグナルのためのシグナル処理を設定します。
   */
  from_iso8601_time(reminders[0].run_time, &run_time);
  remain = run_time - time(NULL);
  alarm((remain > 0) ? remain : 1);

 /* Xt ベースのアプリケーションは、XtAppAddTimeOut を使用して
  * タイマーを設定できます。
   */
  }
}
```

```
/*
 * このルーチンは、ISO 8601 書式の時間表記を、UTC 1970 年 1 月 1 日 00:00:00
 * からの経過秒数を示すティック数へ変換します。
 * このティックはローカル時間に調整されます。
 */ int from_iso8601_time(char *buf, time_t *tick_out)
{
 int  year, month, day, hour, min, sec;
 struct tm time_str;

  sscanf(buf, ``%4d%2d%2dT%2d%2d%2dZ'',
    &year, &month, &day, &hour, &min, &sec);

  time_str.tm_year = year - 1900;
 time_str.tm_mon  = month - 1;
 time_str.tm_mday = day;
 time_str.tm_hour = hour;
 time_str.tm_min  = min;
 time_str.tm_sec  = sec;
 time_str.tm_isdst = -1;
  *tick_out = mktime(&time_str);

  if (*tick_out != (long)-1) {

 /* ローカル・タイムゾーンに調整 */
  if (time_str.tm_isdst == 0)
   *tick_out -= timezone;
  else
   *tick_out -= altzone;
   return(0);
 } else
  return(-1);
}

/*
 * このルーチンは、UTC 1970 年 1 月 1 日 00:00:00 からの経過秒数を示すティック数を、
 * ISO 8601 書式の時間表記へ変換します。
 */
int
to_iso8601_time(time_t tick, char *buf_out)
{
 struct tm time_str;
  if (gmtime_r(&tick, &time_str)) {

 /* 書式文字列が固定幅（ゼロで埋め込み）フィールドを強制する。 */
  sprintf(buf_out, ``%04d%02d%02dT%02d%02d%02dZ'',
   time_str.tm_year + 1900,
   time_str.tm_mon + 1,
   time_str.tm_mday,
   time_str.tm_hour,
   time_str.tm_min,
   time_str.tm_sec);
   return (0);
 } else {
  return (-1);
 }
}
```

用語集

app-defaults ファイル
: プログラマが X リソースを定義するのに使用する、各アプリケーションごとのファイルです。

CDE
: UNIX 上で実行するグラフィカル・ユーザ・インタフェースです。Common Desktop Environment (共通デスクトップ環境) を略したものです。

DATA_HOST
: データ型が読み込まれるホスト・システムを示す DATA_ATTRIBUTESエントリに追加される属性です。データベースが読み込まれるとき以外は、この値を *.dt ファイルに設定しないように注意してください。

IS_ACTION
: アクションが読み込まれるときに作成される DATA_ATTRIBUTESエントリに追加される属性です。DtDtsDataTypeIsAction はこの属性を使用して、データ型がアクション・テーブルから作成されたかどうかを判別します。*.dt ファイルには、アクション・エントリ以外何も表示されません。この値は *.dt ファイルには設定されず、内部的にのみ使用されるので注意してください。

IS_SYSTHETIC
: アクションを読み込まれるときに作成される DATA_CRITERIA/DATA_ATTRIBUTES エントリに追加される属性です。*.dt ファイルには、アクション・エントリ以外は何も表示されません。この値は *.dt ファイルには設定されず、内部的にのみ使用されるので注意してください。

ITE
Internal Terminal Emulator (内部端末エミュレータ) です。ITE によりビットマップ・ディスプレイを端末として使用できます (ログイン画面からコマンド行モードを介して使用します)。

アイコン
画面に表示されるグラフィック記号です。特定の関数またはアプリケーション・ソフトウェアで動作するよう選択できます。

アイコン化
ウィンドウをアイコンにすることです。ウィンドウをアイコン化するプッシュ・ボタンは、ウィンドウ枠の右上角にあります。

[アイコンのインストール**]**
ドラッグ&ドロップによってデスクトップへアイコンをインストールできるようにするサブパネルの選択項目です。

アイテムヘルプ
特定のコマンド、オペレーション、ダイアログ・ボックス、コントロールに関する画面情報をアプリケーションが提供するときのヘルプの書式です。

アクション
アプリケーションが、何らかのオペレーションを実行するために、ファイルのデータベースに定義されたユーザ・インタフェースです。

アクション・アイコン
ファイル・マネージャまたはアプリケーション・マネージャ・ウィンドウで、アクションを表すアイコンです。実行形式ファイルを作成することによってアクション・アイコンを表示し、アイコンが示すアクションと同じ名前を付けます。

アクション・サーバ
アクションの集合へアクセスをサービスするホスト・コンピュータです。

アクティブ
キーボードとマウス入力によって現在影響を受けているウィンドウ、ウィンドウ要素、またはアイコンです。アクティブなウィンドウは、特有のタイトル・バーのカラーや網掛けによってワークスペースにある他のウィンドウとは区別されます。アクティブなウィンドウ要素は、強調表示または選択のカーソルによって示されます。

値
Bento 型コンテナでは、オブジェクトの内容を参照します。「メタデータ」も参照してください。

アタッチメント
ドキュメント内でカプセル化されたデータ・オブジェクトです。

アプリケーション・グループ	特定のソフトウェア・アプリケーションを保持するアプリケーション・マネージャのコンテナです。
アプリケーション・サーバ	アプリケーション・ソフトウェアへアクセスを提供するホスト・コンピュータです。
アプリケーション・マネージャ	ツールやその他の使用可能なアプリケーションを管理するアプリケーションソフトウェアです。
ウィンドウ	ディスプレイ上の矩形の領域です。アプリケーション・ソフトウェアには通常、ダイアログ・ボックスと呼ばれる副ウィンドウを開くことができる主ウィンドウがあります。
ウィンドウ・アイコン	アイコン化されたウィンドウです。
ウィンドウ・メニュー	ウィンドウ・メニュー・ボタンを選択することにより表示されるメニューです。メニューは、[移動]、[サイズ]、[アイコン化]、[最大表示] など、ウィンドウの位置やサイズを指定する選択項目を提供します。
ウィンドウ・メニュー・ボタン	ウィンドウの左上隅にあるコントロールで、タイトル・バーの横にあります。このボタンを選択すると、ウィンドウ・メニューが表示されます。
ウィンドウ・リスト	アクションが選択されたウィンドウに関連付けられているすべてのウィンドウのリストを表示するアクションです。
ウィンドウ枠	アプリケーション・ソフトウェアを囲むウィンドウの可視部分です。ウィンドウ枠には、タイトル・バー、サイズ変更境界、アイコン化ボタン、最大表示ボタン、およびウィンドウ・メニュー・ボタンという最大 5 個のコントロールを入れることができます。
オプション	実行するコマンドを選択するとき、またはダイアログ・ボックスで項目を選択または入力するときに使用できるバリエーションを示す一般用語です。
カーソル	マウスまたはキーボードでの入力によって影響を受ける現在のオブジェクトを示すグラフィカル・デバイスです。

改行文字	ドキュメント内のテキスト行の最後をマークする目に見えない文字です。この文字はプリンタや画面で改行したり、新しい行を始めたりするように指示します。
階層メニュー	他の画面要素と対話するために選択する追加要素を表示するメニュー項目です。
階層リスト	他の画面要素と対話するために選択する追加要素を表示するリスト・ボックスです。
画面ロック	有効なユーザ・パスワードが入力されるまでの間、追加の入力を除外して、ワークステーションの画面をロックする機能です。
キーの割り当て	キー・ストロークと特定の動作を関連付けるものです。
切り替える	マウスかキーボードを使用して、ラジオ・ボタンやチェック・ボックスなど、2 つの状態があるコントロールの状態を変更することです。
クライアント	ネットワーク・サーバからサービスを受けるシステムまたはアプリケーション・ソフトウェアです。
グラブする	マウス・ポインタをオブジェクト上に移動させ、オブジェクトを移動させるためにマウス・ボタン 1 を押し続ける操作です。「ドラッグする」と「ドロップする」を参照してください。
グラブ・ハンドル	選択されたグラフィック・オブジェクトの隅と中心点に表示される小さい正方形です。
クリック	マウス・ポインタを移動しないでマウス・ボタンを押して離すことです。
クリップボード	最後にカット、コピー、またはペーストされたデータまたはオブジェクトを一時的に格納するバッファです。
グループ・ボックス	コントロールのセットを視覚的に関連付けるウィンドウのボックスです。
現在の項目	リストで現在強調表示されている項目です。

現在のセッション　ログアウト時にセッション・マネージャによって保存されるセッションです。次のログイン時に、他のものを指定しなければ、このセッションが自動的に開き、前回の終了時の状態から作業を継続できます。「ホーム・セッション」も参照してください。

現在の設定　チェック・ボックスやラジオ・ボタンなどのコントロールの現在の状態です。

現在のワークスペース　画面に現在表示されているワークスペースです。ワークスペース・スイッチで変更できます。

合成データ属性　アクションをロードしたときに作成される DATA_CRITERIA/DATA_ATTRIBUTES エントリに追加されるデータ型です。`*.dt` ファイルには、アクション・エントリ以外は何も表示されません。この属性は `*.dt` ファイルには設定されず、内部的にのみ使用されるので注意してください。

項目　リスト内の要素です。

コマンド行プロンプト　コンピュータがコマンドを受け付ける準備が整ったことを示すプロンプト (通常は `%`、`>`、または`$`) です。端末エミュレーション・ウィンドウで [Return] キーを押すと、コマンド行プロンプトを表示できます。

コンテキスト・ヘルプ　カーソルまたはポインタが指す特定の選択項目またはオブジェクトに関するヘルプ情報です。

コントロール　アクションを実行したり、オプションの設定を示したりする、さまざまなオブジェクト (ボタン、チェック・ボックス、スクロール・バーなど) の一般用語です。[フロントパネル] アイコンにも適用されます。

コンボ・ボックス　テキスト・ボックスを使用して、コンボ・ボックスのテキスト・ボックス位置を参照し、リスト・ボックスを使用してリスト位置を参照します。たとえば、ファイル・テキスト・ボックスにファイル名を入力したり、下にあるリスト・ボックスから選択したりします。

サーバ　クライアントにサービスを提供するシステムです。

サイズ変更ハンドル　ウィンドウまたはウィンドウの区画のサイズを変更するのに使用するコントロールです。

サイズ変更ポインタ　ウィンドウなどのオブジェクトのサイズ変更中に表示されるマウス・ポインタです。

作業領域　コントロールとテキストが表示されるウィンドウの一部です。

指す　特定の画面のオブジェクトまたは領域上で、ポインタが停止するまでマウスを移動させることです。

サブパネル　追加のコントロールを提供するフロントパネルのコンポーネントです。サブパネルには通常、関連するコントロールのグループが入っています。

実行ホスト　アクションによって起動されるアプリケーションを実行するホスト・コンピュータです。アクションが存在するのと同じコンピュータか、ネットワーク上の別のコンピュータです。

修飾キー　このキーを押しながら他のキーを押すと、第 2 のキーの意味を変更するキーです。[Control]、[Alt]、[Shift] キーなどがその例です。

状態インジケータ　ドロップ領域が有効か無効かを示すフィードバックと結び付けられた位置付け用ポインタとして使用されるドラッグ・アイコンの一部です。

ショートカット　ダイアログ・ボックスへの指定を簡単にするマウス・アクションの一般用語です。ショートカットとしては、ファイル名リスト・ボックスにある項目をダブルクリックすると、1 回のアクションでその項目を選択し、[了解] を選択したことになります。

ショートカット・キー　メニュー・コマンドを起動するのに使用するキーボードのキー・シーケンスのことです。これは、特殊なアクセラレータ・キーを使用するキー・シーケンスや、下線が引かれた文字 (ニーモニック) シーケンスになります。たとえば、[Alt] + [F4] キーか [Alt] + [F] + [P] キーを押すと、コマンドとして [ファイル] => [印刷] を選択したことになります。

書式　CDE ドキュメント・コンテナで、属性を格納するのに使用する型です。

シンボリック・リンク	ファイルまたはディレクトリへの参照です。
スクリーン・セーバ	指定された時間後にワークステーションがディスプレイをスイッチ・オフにする、または表示されるイメージを変更するように指定する選択項目で、これによりディスプレイの寿命が延びます。
スクロール・バー	ウィンドウの右側または下部にあるコントロールで、これにより現在は見ることができないウィンドウの内容を表示できます。
スケーラブル・タイプフェース	特定のサイズ、傾斜、または線の太さのビットマップ・フォントを作成できるタイプフェース用の数学的アウトラインです。
スタイル・マネージャ	(カラー、フォント、キーボード、マウス、ウィンドウ、セッション起動の動作などを含む) ワークスペース環境の視覚的要素とシステム・デバイス動作の一部をカスタマイズするのに使用するソフトウェア・アプリケーションです。
スピン・ボックス	相互に排他的な選択は例外として、曜日など関連のあるもののセットを表示するテキスト・ボックスと 2 つの矢印ボタンが付いたウィンドウ要素です。
スライダ	使用可能な範囲の中から値を設定するためにトラックとアームを使用するコントロールです。アームの位置 (または独立したインジケータ) により現在の設定値がわかります。
セーブバック	変更されたデータを変更前のファイルに書き戻すためにドラッグ&ドロップが機能することです。
セーブバックなし	バッファに保持されたデータの変更を変更前のファイルに書き戻すためにドラッグ&ドロップが機能しないことです。
セッション	ユーザのログインからログアウトまでの経過時間です。
セッション・サーバ	ネットワーク・セッションを提供するシステムです。セッション・ファイルはセッション・サーバにあり、ネットワーク上のシステムにログインするときには必ず使用されます。
選択する **(choose)**	マウスやキーボードを使用して、コマンドかアクションを開始するメニュー項目、ボタン、またはアイコンを選択することです。「選択する (select)」も参照してください。

選択する **(select)**
: オブジェクトを操作したり使用可能にしたりするように、強調表示やその他の目印をオブジェクトに追加することです。選択には、リスト内の項目の強調表示やチェック・ボックスをオンに切り替えるなど、状態の変更以外のアクションの初期化は含まれません。

操作インジケータ
: ユーザに操作 (移動、コピー、またはリンク) のフィードバックを指定するドラッグ・アイコンの一部で、ドラッグ中に示されます。

挿入ポイント
: キーボードに入力されたデータ、またはクリップボードかファイルからペーストされたデータが画面に表示されるポイントです。テキスト入力領域では、カーソルの同義語です。

ソース・インジケータ
: ドラッグされている項目を表すドラッグ・アイコンの一部です。

[属性**]**
: 日時や名前などのオブジェクトの特性の設定を可能にするメニュー・コマンド、またはタイプフェースなどのオブジェクトの特性を識別するディスプレイです。

ソフトウェア・アプリケーション
: 動作するツールを提供するコンピュータ・プログラムです。ソフトウェア・アプリケーションの例としては、スタイル・マネージャ、テキスト・エディタ、ファイル・マネージャなどがあります。

ダイアログ・ボックス
: アプリケーションによって表示される、ユーザによる入力を要求するウィンドウです。簡略形としてダイアログを使用しないでください。

タイトル・バー
: ウィンドウの最上部の領域で、ウィンドウ・タイトルが入っています。

タイル
: パターンまたはビジュアル構造で表面をカバーするのに使用する矩形の領域です。たとえば、ワークスペース・マネージャがタイリングをサポートしている場合、システム・カラーの使用を制限されているユーザは、既存のカラーをブレンドして新しいカラー・タイルを作成できます。

ダブルクリック
: マウス・ポインタを移動せずにマウス・ボタンを 2 回すばやく押すことです。特に明示しなければ、ボタン 1 をクリックしてください。

端末エミュレータ　実行中の非ウィンドウ・プログラムに対して特定のタイプの端末をエミュレートするウィンドウです。端末エミュレータ・ウィンドウは、通常はコンピュータのオペレーティング・システムとの対話のためにコマンドを入力するのに使用します。

チェック・ボックス　設定がチェック・マークの有無で示される非排他的コントロールです。チェック・ボックスには、オンとオフの 2 つの状態があります。

ディスプレイに依存しないセッション　画面の解像度またはカラー機能にかかわらず、任意のディスプレイ上に復元できるセッションです。

ディスプレイに依存するセッション　特定のディスプレイ上にのみ復元できるセッションです。

ディレクトリ　ファイルと他のサブディレクトリの集合です。

データ型　特定のデータ・ファイルを適切なアプリケーションおよびアクションに関連付けるのに使用される機能です。データ型は、特定の拡張名などのファイルの命名規則またはファイルの内容に基づいて、ファイルのタイプを決定できます。

データベース・ホスト　アクションが定義されているホスト・コンピュータです。

データ・ホスト　アクションのデータが位置付けられているホスト・コンピュータです。

テキスト・フィールド　ウィンドウ内の情報が入力される矩形の領域です。キーボード・フォーカスが付いたテキスト・フィールドは、点滅するテキスト挿入カーソルが入っています。

デフォルト　アプリケーションによって自動的に設定される値です。

ドラッグする　マウス・ポインタをオブジェクト上で移動する操作です。マウス・ボタン 1 を押したまま、マウス・ポインタおよびオブジェクトをワークスペースの他の場所へ移動します。

ドラッグアンダ・フィードバック　ドロップ領域で使用される外観のことです。フィードバックはサイトの周りを描画する実線、ドロップ領域の周りに付けた射影によって凹凸に見える表面、ドロップ領域の上に重ねて描画されるピックスマップのいずれかです。

ドラッグオーバ・フィードバック	潜在的なドロップ領域でユーザがドラッグしたときにドラッグ・アイコンの外観が変わることです。
ドラッグ&ドロップ	他のどこかにあるオブジェクトを移動して配置するためにポインティング・デバイスを使用することにより、オブジェクトを直接操作することです。
ドロップする	オブジェクトをグラブしてから、マウス・ボタンを離す動作のことです。オブジェクトが適切な領域にドロップされると、アクションが開始されます。「グラブする」を参照してください。
ドロップ・ターゲット	アプリケーション内のドロップ領域を表す矩形のグラフィックです。
ドロップ領域	[ごみ箱]、[プリンタ]、および [メール・プログラム] アイコンを含むワークスペースの領域で、ドロップされたオブジェクトを受け入れます。オブジェクトは、クイック・アクセスのためワークスペースにドロップできます。
ナビゲーション・キー	カーソルの現在の位置を移動するのに使用するキーボードのキーです。これらには、矢印キー ([Control] キーを一緒に押す場合と押さない場合があります)、[Tab] キー ([Control] キーまたは [Shift] キーを一緒に押す場合と押さない場合があります)、[Begin] キーと [End] キー ([Control] キーを一緒に押す場合と押さない場合があります)、および [Page Up] キーと [Page Down] キーも含まれます。
ネットワーク・セッション	複数のシステムに渡って管理されるセッションです。ネットワーク・セッションを使用すると、ログインするのにどのシステムを使用したかにかかわらず、同一のセッションを参照できます。また、複数のシステムに渡る単一のホーム・ディレクトリも提供します。
ハイパーリンク	ヘルプ・テキストで、選択時に別のヘルプ・トピックを表示する情報です。
バックグラウンド	ボタン、リストなどのオブジェクトが表示されるウィンドウの基本となる領域です。
離す	マウス・ボタンまたはキーボード・キーを離すことです。
パレット	使用可能な要素の範囲で、通常はカラーです。

引き数	コマンドに続く情報の項目です。
ビジー・ポインタ	アプリケーションがビジーで入力を受け付けられないときに表示されるマウス・ポインタです。
ピックスマップ	ラスタ形式で格納されたイメージです。通常、3 色以上を使用したイメージを指します。「ビットマップ」も参照してください。
ビットマップ	ラスタ形式で格納されたイメージです。通常、2 色 (フォアグラウンドとバックグラウンドのカラー) のイメージだけで表されます。「ピックスマップ」も参照してください。
ビットマップ・フォント	ビットマップ・フォントは、ドットのマトリックスで構成されています。「フォント」を参照してください。
[表示オプションのコピー]	現在の表示の属性をコピーし、コピーをクリップボードに置くメニュー・コマンドです。
ファイル・サーバ	アプリケーションが使用するデータ・ファイルを格納するホスト・コンピュータです。
ファイル・マネージャ	システム上のファイルとディレクトリを管理するソフトウェア・アプリケーションです。
フィールド	名前フィールドや電話番号フィールドのようにデータを保持するウィンドウ要素です。[名前] テキスト・ボックスや [ファイル] リスト・ボックスなどのように、なるべく要素を説明するのに特有の名詞を使用してください。
フォアグラウンド	ウィンドウの内容とウィンドウのバックグランドを区別するのに使用するカラーまたは網掛けです。
フォーカス	キーボードによる入力が受け付けられる場所です。
フォルダ	「ディレクトリ」を表すアイコンです。
フォント	1 つのサイズかつ 1 種類のタイプフェースの文字 (英字、数字、および特殊文字) の完全なセットのことです。フォントの例としては、「10 ポイントの Helvetica ボールド」があります。
複合ドキュメント	1 つ以上のアタッチメントを含むドキュメントです。

プッシュ・ボタン　選択するとすぐにアクションを開始するコントロールです。通常ダイアログ・ボックスに見られるプッシュ・ボタンの例には、[了解]、[取消し]、[ヘルプ] などがあります。

プリント・サーバ　1 台以上のプリンタが接続されるホスト・コンピュータ、またはそれらのプリンタを管理する UNIX プロセスです。

フロントパネル　アプリケーションへアクセスするコントロールおよびユーティリティを格納するウィンドウで中央に配置されます。また、ワークスペース・スイッチも格納しています。フロントパネルはすべてのワークスペースに配置されます。

ページ　ウィンドウ内に表示されたテキストを全画面単位に先に進むことで、通常はスクロール・バーを使用します。

ポインタ　現在のマウスの位置と、設定によってはアクティブなウィンドウを示す矢印またはその他のグラフィカル・マーカです。「カーソル」も参照してください。

ホーム・セッション　現在のセッション以外に、次のログイン時に自動的に戻るセッションとして特定のセッションを指定するログアウト時の選択項目です。

ホーム・ディレクトリ　個人用ファイルとディレクトリを保持しておくディレクトリです。デフォルトでは、[ファイル・マネージャ] ウィンドウと端末エミュレータ・ウィンドウを最初に開いた場所がホーム・ディレクトリに設定されます。

ボタン　アプリケーションがアクションを起動するためのウィンドウ・コントロールの一般用語です。通常はコマンドの実行、ウィンドウの表示、メニューの表示を行います。マウスのコントロールの説明にも使用します。

ボタンの割り当て　特定の動作とマウス・ボタンの操作を関連付けたものです。

マッピング　それ自体は実行文字列を持たずに他のアクションを実行するアクションです。ファイル `/usr/vue/types/user-prefs.vf` には、組み込みマップ・アクションがあります。たとえば、フロントパネルで使用する組み込み CDE メール・アクションは、[メール・プログラム] アクションにマップされます。

メタデータ	Bento 型コンテナでは、オブジェクトに関する情報です。「値」も参照してください。
メニュー	特定のアプリケーション・タスクを実行するために選択するコマンドのリストです。
メニュー項目	メニューに表示される選択項目です。
メニュー・バー	メニュー名がリストされるタイトル・バーと作業領域の間にあるアプリケーション・ウィンドウの一部です。
矢印キー	キーボードにある 4 つの方向を示すキーです。「ナビゲーション・キー」も参照してください。
要素	リスト内の項目やウィンドウ内のコントロールなど、より明白なコンテキストにあるスタンドアロン項目と見なすことができるエンティティの一般用語です。
ラジオ・ボタン	その設定がグラフィカル・インジケータの有無によって示される排他的コントロールで、通常はラジオ・グループの一部になります。ラジオ・ボタンには、オンとオフの 2 つの状態があります。
ラジオ・ボタン・グループ	独特のラベルを持つことができるラジオ・ボタンのセットが入っているボックスです。一度に起動できるラジオ・ボタンの数は 1 つです。
ラベル	要素名で、ウィンドウ要素の横に表示されているテキストです。
リスト	選択する要素が入っているコントロールです。選択リストとも呼ばれます。
リスト・ボックス	1 つ以上の項目を選択できるような項目のリストを表示する、任意の数のグラフィカル・デバイスです。通常は使用するボックスの種類を指定する必要はありません。
リソース	ウィンドウまたはアプリケーションの属性 (外観または動作) を指定する X Window System の機構です。リソースには、コントロールする要素にちなんだ名前が通常付けられます。
リンク	シンボリック・リンクの同義語です。

ローカル・ホスト	ソフトウェア・アプリケーションを実行している CPU またはコンピュータです。お手元のワークステーションのことです。
ワークスペース	現在の画面ディスプレイや、そのディスプレイにあるアイコンとウィンドウ、およびオブジェクトを位置付けることができる未使用の画面領域です。
ワークスペース・オブジェクト	ファイル・マネージャからワークスペースにコピーされたオブジェクトです。
ワークスペース・スイッチ	いくつかのワークスペースから 1 つだけ選択できるようにするコントロールです。
ワークスペース・バックグラウンド	ディスプレイの中で、ウィンドウ、アイコン、またはオブジェクトには覆われていない部分です。
ワークスペース・マネージャ	複数のワークスペース内のウィンドウのサイズ、位置、および操作をコントロールするソフトウェア・アプリケーションです。ワークスペース・マネージャには、フロントパネル、各アプリケーションを囲むウィンドウ枠、および ウィンドウ・メニューと [ワークスペース] メニューが含まれます。
[ワークスペース]メニュー	ワークスペースの何もない領域を指し、マウス・ボタン 3 をクリックすることによって表示されるメニューです。

索引

せ

そ

て

と