TR-I-0092

X Window System, Version 11 ポケット・ガイド
- X Toolkit 編 -
X11 Pocket Guide - X Toolkit -

田中 孝明
Takaharu TANAKA

1989 . 8

概要

本稿はプログラマ向けに「X Window System, Version 11 ポケット・ガイド」として作成したテキストを再構成したものである。

MIT で開発された X Window System は VAX をはじめ多くのコンピュータ上にインプリメントされ、事実上、グラフィック/ウィンドウ・システムの業界標準となっている。

X Toolkit は Xlib の上位に位置するライブラリ・ルーチンで、これを用いることによって、より高度なユーザ・インタフェースを構築することができる。本稿では標準である X Toolkit に加えて、DEC の提供するより完成された XUI Toolkit にも言及する。

本稿が研究支援システムおよびデモンストレーション・システムを開発する際の一助となれば幸いである。なお、本稿は「X Window System, Version 11 ポケット・ガイド - Xlib 編 - 」(TR-I-0091) との二部構成となっている。

ATR 自動翻訳電話研究所
ATR Interpreting Telephony Research Laboratories

Table of Contents

Chapter 3

X Toolkit

3.1. X Toolkit とは

X Toolkit は X Window System の Xlib 上に作られたサブルーチン・ライブラリです。X Toolkit は Xlib の機能と操作性を拡張し，高度なユーザ・インターフェースとプログラム・インターフェースを提供します。

ユーザから見ると X Toolkit を用いたアプリケーションは，ラベルやボタン，スクロール・バーなど一連の部品（widget）を組み合わせたものとなります。プログラムからはこの部品は内部データとそのデータに対する処理ルーチン，および表示ルーチンを持った一種のモジュールとして扱われます。そして，これらの部品はサーバから送られてくるイベントの処理や，各部品の大きさや位置関係の管理（ジオメトリ・マネージメント），部品の持つ内部データの管理（リソース・マネージメント）などを内部ルーチンによって自動的に行なうようになっています。

3.1.1. X Toolkit の構造

X Toolkit は一連の widget（部品）と widget を構築するための intrinsic 関数からなっています。intrinsic は一つ一つの widget を作成，表示したり，サーバからのイベントを対応する widget に振り分けたりする関数群です。一方の widget は自分自身がどういうデータを持ち，それをどう処理するのかの定義を持っています。

intrinsic は widget の実現にオブジェクト指向を取り入れています。オブジェクト指向というのは内部データとそのデータに対する操作を一種のモジュール（オブジェクト）の中に閉じ込めることによって，プログラムの信頼性，可読性，保守性を向上させようという，構造化言語を一歩進めた考え方です（Smalltalk-80, Symbolics Common Lisp の flavor, C++ などで採用されている）。オブジェクト内のデータはまた，そのオブジェクトが外部に対して提供する手続き (method) によってのみ操作することができます。各オブジェクトのふるまいはそのオブジェクトの属するオブジェクト・クラスによって定義されます。オブジェクト・クラスは階層構造を持ち，上位のクラス（スーパー・クラス）の属性を下位のクラスが受け継ぐことができるようになっています（インヘリタンス：継承）。このため，複数のオブジェクトが持つ共通の処理をそのクラスや上位のクラスなど一箇所で定義するだけですみます。

X Toolkit の intrinsic は次の機能を提供します。

- o X Toolkit の初期化
- o widget の作成，起動，削除

- o ウィンドウ・イベントおよびファイル・イベント，タイマ・イベントの管理
- o widget のジオメトリ管理
- o インプット・フォーカス
- o widget のリソース管理（リソース・マネージメント）
- o イベントのトランスレーション（トランスレーション・マネージメント）
- o エラー処理

3.1.2. widget

X Toolkit の widget は見た目には，いくつかのウィンドウとそこに表示されるラベルやアイコンで構成されています。そして，マウスやキーボードの操作によって widget 毎に決められたある一定の動作を起こします。

widget はその内部に自分自身に必要なすべてのデータとそのデータに対する操作のための手続きを持っています。widget の外からは直接これらのデータを操作することはできません。

widget の動作と内部のデータはその widget の属する widget クラスによって定義されます。widget クラスは階層構造を持っていて，上位のクラス（スーパー・クラス）の定義は下位のクラスに受け継がれます。また，widget クラスの定義に基づいて作成された一つ一つのオブジェクトを widget インスタンスと呼びます。同じクラスに属するインスタンスは動作の定義は共有しますが，個々のデータは共有しません。また，widget インスタンスもウィンドウと同様の親子関係を持ちます。

widget クラスと widget インスタンスはそれぞれ名前を持ちます。一般的にクラス名の頭文字は大文字，複合語では後ろの単語の頭文字を大文字とします（Command クラス，TopLevelShell クラスなど）。これらの名前は widget インスタンスのリソースを指定，検索するために用います。

widget インスタンスは必要な内部データをリソース・リストという形で保持しています。プログラムからリソースを指定する場合にはリソース名に対応したシンボルを用いて ArgList という形で行ないます。このシンボルはリソース名の前に XtN をつけた形になっています（XtNwidth, XtNlabel など）。widget のリソースをリソース・ファイルで指定する場合は

```
xedit*Command.background :  yellow
xedit*Quit.background :     red
xedit*font :                vtsingle
```

などのようにリソース名と widget のインスタンス名，クラス名を組み合わせて指定することができます。

マウス・ボタンのクリックなど各 widget で定められたユーザの操作によって，アプリケーションの処理（関数）を起動させることもできます。このためには widget に対してコールバック・プロシージャを定義してやります。コールバック・プロシージャとして定義された処理はユーザの操作や widget 内部の状態の変化によって自動的に widget から呼び出されます。X Toolkit を用いたアプリケーションの主要な処理はこのコールバック・プロシージャの形で記述します。

widget は widget ID で識別します。widget ID は実際には widget のインスタンスの構造体へのポインタとなっていますが，アプリケーションからはこれを単に ID として用います。widget ID からその widget を形作っているウィンドウのウィンドウ ID や，そのウィンドウの存在するディスプレイ，スクリーンなどを求める関数などが用意されています。

3.2. X Toolkit を用いたプログラムの流れ

X Toolkit を使うとプログラムは次のような流れとなります。

1. X Toolkit のヘッダファイル（<X11/Intrinsic.h> と <X11/StringDefs.h>）と各 widget のヘッダファイル（<X11/Label.h> など）のインクルード（必要なら <X11/Xatom.h> と <X11/Shell.h> もインクルード）
2. X Toolkit の初期化
3. widget インスタンスの作成
   i. コールバック・プロシージャの作成および宣言（アプリケーションの主要な処理はここで行なう）
   ii. 引き数リスト（ArgList）の作成
   iii. XtCreateWidget の呼び出し
   iv. ペアレント widget への登録
4. widget の起動
5. イベントのディスパッチ

各 widget のヘッダファイルはプログラムで使用している widget のものをインクルードします。ヘッダファイルの名前は一般的に <X11/Classname.h> となっています。

Chapter 4

X Toolkit の基本関数

アプリケーションで用いる X Toolkit intrinsic の最も基本的な関数を挙げます。

4.1. X Toolkit の初期化

X Toolkit を使用するには，まずその初期化を行なう必要があります。

```
Widget XtInitialize (shell_name, application_class,
                     options, num_options, argc, argv)
    String  shell_name;
    String  application_class;
    XrmOptionDescRec  options[];
    Cardinal  num_options;
    Cardinal  *argc;
    String  argv[];
```

XtInitialize は X Toolkit を初期化すると同時に，リソース・データベースのロード，コマンド行の解析，およびトップレベル widget の作成を行ないます。

shell_name はトップレベル widget につけるインスタンス名です。application_class はアプリケーションが属するクラスの名前です。このクラス名は通常，アプリケーションの名前の最初の 1 文字か 2 文字を大文字にしたものを用います。アプリケーション名にはリソースによる指定がなければ argv[0] の最後の部分が使われます。

options はコマンド行をどのように解析するかを指定するものです（後述）。argc, argv はアプリケーションを起動した際のコマンド行の引き数で，main 関数のパラメータをそのまま指定します。X Toolkit はこれを options に従って解析します。その結果リソース・マネージャで使った引き数は argc, argv から取り除かれます。

XtInitialize によって作られるトップレベル widget はシェル widget クラスのインスタンスとなっています。シェル widget というのはアプリケーションのトップレベルのウィンドウを構成する特別の widget です。シェル widget はアプリケーションとウィンドウ・マネージャとのデータのやり取りを専属して行ないます。ウィンドウ・マネージャによるアプリケーションウィンドウのリサイズやアイコンの設定などもシェル widget によって処理されます。また，アプリケーションで用いる他のすべての widget はこの widget のサブ widget として作られていきます。

4.2. widget インスタンスの作成

widget インスタンスは次の関数で作成します。

```
Widget XtCreateWidget (name, widget_class, parent, args, num_args)
   String  name;
   WidgetClass  widget_class;
   Widget  parent;
   ArgList  args;
   Cardinal  num_args;
```

name は widget のインスタンス名，widget_class は widget のクラスです。parent にはペアレント widget の widget ID を指定します。ArgList は widget インスタンスのリソースの値を指定するための構造体 Arg の配列です。これは

```
typedef struct {
   String  name;
   XtArgVal  value;
} Arg, *ArgList;
```

という形をしています。name メンバにリソースの名前（XtNlabel など）を，value メンバにその値を設定します。この ArgList を作るために XtSetArg というマクロが用意されています。

```
#define XtSetArg(arg, n, d) \
    ( (arg).name = (n), (arg).value = (XtArgVal)(d) )
```

これを用いると次のようにして ArgList を作ることができます。

```
Arg  arglist[10];
int  i=0;
XtSetArg (arglist[i], XtNwidth, 400);  i++;
XtSetArg (arglist[i], XtNheight, 300);  i++;
```

XtCreateWidget の num_args にはここで計算した i を指定します。XtSetArg はマクロとして定義されているので i++ は外側で行なわないといけません。また，

```
static Arg  arglist[] = {
   {XtNwidth,  (XtArgVal) 400},
   {XtNheight,  (XtArgVal) 300},
};
```

という様に ArgList を作り，num_args に XtNumber (arglist) を指定してやることもできます。 XtNumber は #define... で ArgList などの配列の大きさを求めるマクロです。

ArgList は widget 作成時に指定する他に

```
void XtSetValues (w, args, num_args)
   Widget  w;
   ArgList  args;
   Cardinal  num_args;
```

で設定することもできます。また，

```
void XtGetValues (w, args, num_args)
   Widget  w;
   ArgList  args;
   Cardinal  num_args;
```

で widget のリソースの値を参照することもできます。ただし，XtGetValues の ArgList の value フィールドには値を格納する変数のアドレスを指定します。

4.3. コールバック・リスト

多くの widget はボタンのクリックなどのユーザの操作に対応付けてアプリケーションの処理を起動することができます。widget は他の条件の下，定められたイベントが発生すると，登録されたアプリケーションの関数を呼び出します。この関数をコールバック・プロシージャと呼びます。コールバック・プロシージャの設定は，widget のコールバック・プロシージャ用のリソースに対してコールバック・リストを指定することによって行ないます。コールバック・リストとコールバック・プロシージャは

```
typedef void (*XtCallbackProc) ()

typedef struct {
   XtCallbackProc  callback;
   caddr_t  closure;
} XtCallbackRec, *XtCallbackList;

void callback (widget client_data, call_data)
  Widget  widget;
  caddr_t  client_data;
  caddr_t  call_data;
```

closure, client_data
: アプリケーションがコールバック・プロシージャに渡すデータ。closure で指定したデータがそのまま client_data として渡される

widget
: コールバック・プロシージャを呼び出した widget

call_data
: widget がコールバック・プロシージャに渡すデータ。内容は widget 毎，リソース毎に決まっている

と定義されています。コールバック・リストの最後は NULL でターミネートしておきます。

すでにある widget のコールバック・リストを操作するための関数として

```
void XtAddCallback (w, callback_name, callback, client_data)
   Widget  w;
   String  callback_name;
   XtCallbackProc  callback;
   caddr_t  client_data;

void XtAddCallbacks (w, callback_name, callbacks)
   Widget  w;
   String  callback_name;
   XtCallbackList  callbacks;
```

が用意されています。callback_name には対応する widget のコールバック名（リソース名）を指定します。この他に XtRemoveCallback (...), XtRemoveCallbacks (...), XtHasCallback (widget, callback_name) という関数があります。また，XtSetValues を用いて設定する事もできます。

4.4. ペアレント widget への登録（manage）

作成した widget インスタンスはペアレント widget に登録することによって初めてジオメトリ・マネージメントおよび表示の対象となります。このために次の関数が用意されています。

```
typedef Widget  *WidetList;

void XtManageChildren (children, num_children)
   WidgetList  children;
   Carinal  num_childern;

void XtManageChild (child)
   Widget  child;

void XtUnmanageChildren (children, num_children)
   WidgetList  children;
   Cardinal  num_children;

void XtUnmanageChild (child)
   Widget  child;
```

XtManagerChildren では共通のペアレント widget を持つ widget を WidgetList に指定します。また，

```
void XtCreateManagedWidget (name, widget_class, parent, args, num_args)
   String  name;
   WidgetClass  widget_class;
   Widget  parent;
   ArgList  args;
   Cardinal  num_args;
```

を用いて XtCreateWidget と XtManageChild の処理を一度に行なうこともできます。

4.5. widget の起動（realize）

作成した widget は起動（realize）することによってウィンドウの作成，表示され，イベント処理が始められます。これを行なうには次の関数を用います。

```
void XtRealizeWidget (w)
   Widget  w;
```

XtRealizeWidget は指定した widget とその widget に登録されているサブ widget の起動を行ないます。

4.6. イベントのディスパッチ（メイン・ループ）

X Toolkit を用いたアプリケーションでは widget が順次イベントを処理していくことによって動作します。intrinsic はサーバから送られてくるイベントを widget に自動的に振り分けます。ある種のイベントはアプリケーションが設定したコールバック・プロシージャを起動します。1 つのコールバック・プロシージャの処理が終わると，intrinsic は再びイベントを処理するループを続けます。

このように，サーバで起きたイベントを順次取り込み，それらを対応する widget に振り分け，処理させるには次の関数を用います。

```
void XtMainLoop ()
```

XtMainLoop は次のように定義されています。

```
void XtMainLoop ()
{
        XEvent  event;

        for (;;) {
                XtNextEvent (&event);
                XtDispatchEvent (&event);
        }
}
```

XtDispatchEvent は渡されたイベントに対応するイベント・ハンドラ（後述）を起動し，widget に処理を振り分けます。XtNextEvent は XNextEvent と同様，Xlib のイベント・キューの先頭のイベントを取り出すのに加えて，XtAddInput で指定した X イベント以外の入力ソースと XtAddTimeOut によるタイムアウトもポーリングします。

4.7. リソース・マネージメント

widget はリソースという形で内部データを管理します。リソースはプログラムから，もしくはプログラムの実行時に自由に指定できるようになっています。指定されなかったリソースはデフォルトの値が使われます。

4.7.1. widget のリソース

X Toolkit が widget インスタンスを作る際，その位置や大きさ，色，ラベルなど多くの情報を参照します。これらを widget のリソースと呼びます（ウィンドウ ID, GContext などサーバの管理するリソースとは別のもの）。最終的にこれらの情報は各インスタンスのリソース・リストとして管理されます。

Xlib のリソース・マネージャ（Xrm）はデータを階層的に分類して保守，管理する一種のデータベース・マネージャです。これはちょうど複数のウィンドウ間の関係や widget インスタンス間，widget クラス間の関係を表現するのに適しています。X Toolkit はこのリソース・マネージャを用いて，widget のリソースの管理，検索を行ないます。

アプリケーションおよびそのユーザが widget のリソースを指定するには次の 4 つの方法があります。

1. プログラム中の XtCreateWidget や XtSetValues での指定（ArgList を用いる）
2. リソース・ファイルによる指定
3. アプリケーション起動時のオプション指定
4. widget クラスの持つデフォルトの使用

これらの指定は前者ほど優先されます。また，2 と 3 の指定は XtInitialize を呼び出すと自動的に行なわれます。明示的に指定されなかったリソースは widget クラスで定義されているデフォルトの値が使われます。

4.7.2. ArgList によるリソースの指定

プログラム中でのリソースの指定は ArgList（Arg の配列）を用いて行ないます。

```
typdef struct {
   string  name;
   XtArgVal  value;
} Arg, *ArgList;
```

name メンバにリソースの名前，value メンバにその値を指定します。各リソースの名前に対応したシンボルが <X11/StringDefs.h>（共通のもの）と各 widget のヘッダーファイルで定義されています。これは XtNx, XtNbackground などのようにリソース名の前に XtN をつけた形になっています。ArgList によるリソースの指定は widget インスタンス毎に XtCreateWidget や XtSetValues で行ないます。

4.7.3. リソース・ファイルによるリソースの指定

リソース・ファイルはリソースの指定を Xlib のリソース・マネージャが読める形で記述したテキスト・ファイルです。XtInitialize は内部で XrmGetFileDatabase を呼んでこれらをロードしています。

X Toolkit は次の 4 つのリソース・ファイルを参照します。

- o アプリケーションのクラス・リソース・ファイル
  アプリケーションのクラス毎にリソースの指定を行なうためのもの。/usr/lib/X11/app-defaults/Classname（Classname はクラス名）という名前のファイル。通常，アプリケーションの作成者が作成する

- o アプリケーション毎のユーザ・リソース・ファイル
  ユーザとアプリケーション自身によるアプリケーションのカスタマイズに用いる。XAPPLRESDIR というユーザの環境変数の値にアプリケーションのクラス名を付加したものがファイル名となる。XAPPLRESDIR がない時はユーザのホーム・ディレクトリとなる

- o サーバ・リソース・ファイル
  サーバ（ルート・ウインドウ）の RESOUCE_MANAGER というプロパティの値がそのままその内容となる。このプロパティがない場合，ユーザのホームディレクトリの .Xdefaults の内容が使われる

- o ユーザ環境毎のリソース・ファイル
  ユーザの環境変数 XENVIRONMENT の値がその名前となる。この環境変数がない場合はホームディレクトリの .Xdefaults-host（host はクライアン

トのホスト名）というファイルが使われる

それぞれのリソース・ファイル中のリソースの指定は widget の階層構造に対応付けて行うようになっています。階層の指定には widget のインスタンス名や widget クラス名，またそれらを任意に組み合わせたものを用います。中間の階層を省略して指定することもできます。インスタンス名による階層の指定はクラス名による指定より優先され，詳しい指定は省略した指定より優先されます。

フォーマットは次の通り。

```
resourcename :          valuestring     （もしくは）
name.resoucename :      valuestring     （もしくは）
name*resoucename :      valuestring     （もしくは）
*name.resoucename :     valuestring
```

resourcename はリソース名，valuestring はその値，name は widget のインスタンス名もしくはクラス名

1 行に 1 つの指定を行ないます。"." の指定は前後の name が連続した階層であることを示します。"*" は中間の階層を省略する場合の指定です。"name." と "name*" はいくつも続けたり，組み合わせたりして指定することができます。行頭が "!" の行はコメントです。行末に "\" を書くことによって次の行を継続行とすることができます。

例えば次のようになります。

```
! for all command buttons
*command.font :                 8x13
*command.background :           blue
*Command.foreground :           green
! for xmh
xmh.toc*Command.foreground :    black
xmh*background :                red
```

この指定によって xmh というアプリケーションでは，コマンド・ボタンを含めてバックグランドは red となります。また，toc ボックスのコマンド・ボタンのフォアグランドは black，他のコマンド・ボタンのフォアグランドは green となります。

4.7.4. コマンド行によるリソースの指定

コマンド行のオプション指定によって widget のリソースを変更することもできます。X Toolkit は次のオプションをあらかじめ用意しています。

指定	リソース
-background color	background
-bd color	borderColor
-bg color	background
-borderwidth width	borderWidth
-bordercolor color	borderColor
-bw width	borderWidth
-d display	display（サーバの指定）
-display display	display（サーバの指定）
-fg color	foreground
-fn fontname	font
-font fontname	font
-foreground color	foreground
-geometry geometry	geometry（ウィンドウの大きさ，位置）
-iconic flag	iconic（アイコン状態で起動）
-name name	name（アプリケーション名）
-reverse	reverseVideo（反転表示）
-rv	reverseVideo（反転表示）
+rv	reverseVideo（非反転表示）
-selectionTimeout msec	selectionTimeout（クライアント間の通信のタイムアウト）
-synchronous	synchronize（Xlib の同期処理）
+synchronous	synchronize（Xlib の非同期処理）
-title string	title（アイコンのタイトル）
-xrm "resourcename : string"	リソースを直接指定

XtInitialize は XrmParseCommand によってこれらの指定を解釈します。アプリケーションでこれ以外の指定を用いるためには XrmOptionsDescRec で指定します。XrmOptionsDescRec は

```
typedef struct {
   char           *option;     /* Option abbreviation in argv */
   char           *specifier;  /* Resource specifier */
   XrmOptionKind  argKind;     /* Which style of option it is */
   caddr_t        value;       /* Value to provide if XrmoptionNoArg */
} XrmOptionDescRec;
```

という構造体です。option フィールドがオプション文字列（"-background" など），specifier がそのオプションで設定されるリソース（XtNbackground など）の指定です。argKind でそのオプションの値の指定の方法を指示します。これには

XrmoptionNoArg	値はコマンド行に指定しない。そのオプションの指定自身が意味を持つ（-reverse, -synchronize など）
XrmoptionIsArg	そのオプションの文字列自身が値となる
XrmoptionStickyArg	そのオプションに間を空けずにすぐ続く文字列が値となる
XrmoptionSepArg	そのオプションの次の文字列が値となる（-background）
XrmoptionResArg	次の文字列にリソース名とその値がくる（-xrm）
XrmoptionSkipArg	このオプションと次の文字列をスキップする
XrmoptionSkipLine	このオプションから行の終わりまでをスキップする

があります。XrmoptionNoarg では value メンバにその値を指定しておきます。

4.8. トランスレーション・マネージメント

widget は特別な場合を除き intrinsic のトランスレーション・マネージャを用いてイベントとそれによって引き起こされる動作の対応付けを行なっています。トランスレーション・マネージャは widget クラスの用意したトランスレーション・テーブルに基づいてイベントと widget の動作の対応付けを行ないます。アプリケーション・プログラムとそのユーザは widget インスタンスの translations（XtNtranslations）というリソースに別のトランスレーション・テーブルを指定する事によって，この対応付けの一部もしくは全体を置き換える事ができます。

4.8.1. トランスレーション・テーブルのフォーマット

トランスレーション・テーブルは例えば次のように記述します（Command widget のデフォルトのテーブル）。

```
"<EnterWindow>: Highlight()\n\
 <LeaveWindow>:  UnHighlight()\n\
 <Btn1Down>:     Set()\n\
 <Btn1Up>:       Notify() Unset()"
```

各トランスレーションは "\n"（ニュー・ライン）で区切ります。":" の左側をイベント・シーケンス，右側をアクション・シーケンスと呼びます。

イベント・シーケンスは X のイベントの指定を "," で区切って並べたものです。ここで指定したイベントが起こると右側のアクションが起動されます。イベントの指定は次のようにします。

keychar...　　（または）
modifier_list<event_type>(count)detail

keychar　キーボードで押された文字（KeyPress イベント）の指定。前に "^" を付けるとコントロール・キー，"$" を付けるとメタ・キーを同時に押すことの指定となる。""" や "^", "$", "\" を指定するにはそれぞれの前に "\" を付ける必要がある

modifier_list
キー・イベントやボタン・イベントのモディファイアの指定。何も指定しなければどんなモディファイアが押されていても良いことになる。"None" の指定で全く押されていないことの指定となる。モディファイアの前に "!" を付けると，押されているモディファイアが指定と完全に一致していることが必要となり，付けなければ指定していないモディファイアの状態は感知しない。また，"~" を付けるとそのモディファイアが押されていないことの指定となる。":" を付けると Xlib の標準のモディファイア（Shift と Lock）の状態を考慮した解釈をする（例えば :<Key>A と :<Key>a は別扱い）

<event_type>
X のイベント・タイプの指定。

(count)　指定した回数のイベントが続けて起こった時にアクションを起動する（例えば <BtnUp>(2) でダブル・クリック）。"(count++)" で何回以上という指定になる

detail　キーまたはボタンの指定（イベントの detail）。キーの指定は文字やシンボル（"Return", "Tab" など），文字コード（C 言語にならった 10 進，8 進，16 進の指定）で行なう。シンボルには <X11/keysymdef.h> の中の定義の前に付いている "XK_" を外したものを用いる

モディファイアには次のものを指定することができます。

Ctrl または c, Shift または s, Lock または l, Meta または m, Hyper または h, Super または su, Alt または a, Mod1, Mod2, Mod3, Mod4, Mod5, Button1, Button2, Button3, Button4, Button5, ANY, None

イベント・タイプには次のものを指定することができます（すべてのイベント・タイプに対応している）。

Key または KeyDown, KeyUp, BtnDown, BtnUp, Motion または PrtMoved または MouseMoved, Enter または EnterWindow, Leave または LeaveWindow, FocusIn, FocusOut, Keymap, Expose, GrExp, NoExp, Visible, Create, Destroy, Unmap,

Map, MapReq, Reparent, Configure, ConfigureReq, Grav, ResReq, Circ, CircReq, Prop, SelClr, SelReq, Select, Clrmap, Message, Mapping

アクション・シーケンスは次の様に記述します。

```
name()              （または）
name(param, ...)
```

name　　アクション名。アクション名は各 widget またはアプリケーション・プログラムがあらかじめ登録する

param　　name で指定したプロシージャに渡す引数の指定。文字列として渡される。

各 widget は必要なアクションをあらかじめ用意しています（先の例の HighLight や Set など）。また，複数のアクションをブランクで区切って指定することもできます。

トランスレーション・テーブルで用いるアクション名をアプリケーション・プログラムで登録（追加）するには次の関数を用います。

```
void XtAddActions (actions, num_actions)
   XtActionList  actions;
   Cardinal  num_actions;
```

XtActionList は XtActionsRec の配列で次の様に定義されています。

```
typedef void (*XtActionProc)();

typedef struct _XtActionsRec {
   String  action_name;
   XtActionProc  action_proc;
} XtActionsRec, *XtActionList;
```

action_proc は次の様な形式で呼び出されます。

```
void ActionProc (w, event, params, num_params)
   Widget  w;
   XEvent  *event;
   String  *params;
   Cardinal  *num_params;
```

定義するアクションの名前は widget が用意しているアクションの名前と重なっていてはいけません（置き換えられる）。定義したアクションはそのアプリケーション内であればどの widget からも用いることができます。

4.8.2. トランスレーション・テーブルの登録

トランスレーション・テーブルを widget に登録するにはプログラムから行なう方法とリソース・ファイルで行なう方法の 2 つがあります。

プログラムからトランスレーション・テーブルを登録にはまず，次の関数で文字列形式のトランスレーション・テーブルを intrinsic の処理できる形にコンパイルします。

```
XtTranslations XtParseTranslationTable (table)
   String  table;
```

そして，ここで返される XtTranslations を widget の XtNtranslations（Core widget で定義）に設定します。これによって，その widget のトランスレーション・テーブルは table の指定に置き換えられます。

widget のトランスレーション・テーブルの一部だけを変更したい場合には次の関数を用います。

```
void XtArgumentTranslations (w, translations)
   Widget  w;
   XtTranslations  translations;

void XtOverrideTranslations (w, translations)
   Widget  w;
   XtTranslations  translations;
```

widget のトランスレーション・テーブルで指定されているイベント・シーケンスが引き数で指定したものと重複している場合，XtArgumentTranslations は widget に設定されているものを優先し，XtOverrideTranslations は新しいものを優先します。

リソース・ファイルで指定する場合は例えば次のようにします。

```
*Text.translations:  #replace\n\
        <Key>Right:     forward-character()

*Text.translations:  #argument\n\
        <Key>Left:      backward-character()

*Text.translations:  #override\n\
        Meta<Key>F:     forward-word()
```

リソース値の最初の行に "#" に続けて置き換えの方法を指定します。#replace がデフォルトです。

Chapter 5

widget の内部構造とその定義

5.1. intrinsic の widget

X Toolkit intrinsic は次の widget を定義しています。

Core widget　すべての widget の核となる widget。widget のウィンドウの作成のための機能と widget の本質的な処理を備えている

Composite widget
　他の widget をサブ widget として管理するための基本的な構造を持つ widget。スーパー・クラスは Core

Constraint widget
　Composite widget に加えて，ペアレント widget とサブ widet およびサブ widget 間の関係を表現するための構造を持つ widget。スーパー・クラスは Composite

Shell widget　アプリケーションのトップ・レベル widget のための widget。ウィンドウ・マネージャとの協力してアプリケーションのウィンドウの大きさの制御などを行なう。スーパー・クラスは Composite

5.2. widget の定義

widget はその内部に持つデータとそのデータに対する操作から成っています。widget の定義はこれらを intrinsic が認識できる形で宣言することによって行ないます。

各 widget のインスタンスは内部データをインスタンス・レコードという形で保持します（XtCreateWidget の戻り値の widget ID はこのポインタ）。また，widget インスタンスに対する操作やインスタンス内部のデータに対する処理は，それぞれの widget クラスのクラス・レコード（XtCreateWidget で WidgetClass として指定するもの）として保持されます。

5.2.1. widget のヘッダファイル

widget の操作に必要で，外部から参照されるシンボルや型の宣言は widget のヘッダファイルという形にまとめて行ないます。このヘッダ・ファイルの名前の付け方とその内容は次のように決められています。

1) パブリック・ヘッダファイル
その widget を利用するアプリケーションから引用されるヘッダファイル。ファイル名は Classname.h（Label.h, Command.h など）。内容は

i. その widget のスーパー・クラスのパブリック・ヘッダファイルのインクルード。スーパー・クラスの宣言を引き継ぐ

ii. 新しく追加する（スーパー・クラスにはない）リソースのリソース名とリソース・クラス名の定義（#define）

iii. クラス・レコードの extern 宣言

iv. インスタンス・レコードの構造体の型宣言（typedef）

2) プライベート・ヘッダファイル
その widget やサブ・クラスの widget の定義に用いるヘッダファイル。widget の内部データなどの宣言を行なう。ファイル名は ClassnameP.h（LabelP.h, CommandP.h など）。次のような宣言が含まれる

i. その widget のパブリック・ヘッダファイルのインクルード

ii. その widget のスーパー・クラスのプライベート・ヘッダファイルのインクルード

iii. その widget のインスタンス・レコード（構造体）の個々のフィールドの宣言（typedef struct）

iv. その widget のクラス・レコードの個々のフィールドの宣言（typedef struct）

v. widget で新しく定義した処理をサブ・クラスへ継承させるための宣言

インスタンス・レコードとクラス・レコードへの新しいフィールドの追加は，その widget のスーパー・クラスのものの直後に順次付加する形で行ないます。これはスーパー・クラスの処理をそのままサブ・クラスに適用できるようにするためです。intrinsic と各 widget のルーチンはインスタンス・レコードとクラス・レコードの各フィールドを参照するのに，先頭からのオフセットだけを用いています。

例として Athena Widget の Label Widget のインスタンス・レコード（LabelRec または LabelWidget）の定義（LabelP.h に入っている）を以下に示します。

```
/* New fields for the Label widget record */
typedef struct {
    /* resources */
    Pixel       foreground;
    XFontStruct *font;
    char        *label;
    XtJustify   ustify;
    Dimension   internal_width;
    Dimension   internal_height;
    Pixmap      pixmap;
    Boolean     resize;

    /* private state */
    GC          normal_GC;
    GC          gray_GC;
    Position    label_x;
    Position    label_y;
    Dimension   label_width;
    Dimension   label_height;
    Dimension   label_len;
} LabelPart;

/* full instance record declaration */
typedef struct _LabelRec {
    CorePart    core;
    SimplePart  simple;
    LabelPart   label;
} LabelRec;
```

クラス・レコードの内，Core クラスで定義された部分を Core パート（CorePart），そのクラスで新しく追加した部分をクラス・パートと呼ぶことがあります。

intrinsic は主にクラス・レコードの CoreClassPart とインスタンス・レコードの CorePart を用いて widget の操作を行ないます。このことから widget を定義するための大部分の仕事は CoreClassPart のフィールドを埋めることであるといえます。以下の項では CoreClassPart のフィールドに沿った形で基本的な widget の定義の方法について述べていきます。

5.2.2. widget クラスの初期化のための処理（class_initialize, class_part_initialize, class_inited)

クラス・レコードのほとんどの部分は静的に定義されます（コンパイル時に処理される）。しかし，フォントや GC のアロケート，リソースのタイプ・コンバータの宣言などは実行時でないとできません。class_initialize のプロシージャはこういったデータの内，widget クラス全体で共有するものを初期化するために用います。class_initialize は XtProc という型で，次のように定義します。

```
typedef void (XtProc)();

void Proc ()
```

クラス・レコードの内，クラス・パート・レコードの初期化は class_part_initialize で行ないます。

```
typdef void (XtClassProc)();

void ClassProc (widgetClass)
   WidgetClass  widgetClass;
```

class_initialize と class_part_initialize はそのクラスのインスタンスが初めて作られる時に 1 度だけ呼び出されます。この際，その widget のスーパー・クラスが初期化されていない場合はそれらのものが先に呼ばれます。クラス・レコードの class_inited はそのクラスがすでに初期化されたかどうかを示します（定義時は False にしておく）。また，初期化の操作が必要でない場合は class_initalize や class_part_initialize に NULL を指定して省略することもできます。

5.2.3. widget インスタンスの初期化のための処理（initialize, initialize_hook)

XtCreateWidget によってインスタンスが作成されようとする時，initialize プロシージャと initialize_hook プロシージャが呼び出されます。

initialize プロシージャでは次のことを行ないます。

- o アドレス（ポインタ）で指定されているリソース（文字列など）のための領域のアロケートとその内容のコピー（コールバック・リストは除く）
- o 値の設定されていないリソースへの適切な値の設定（width や height に 0 が指定されている場合など）
- o リソースによって決まるリソース以外のフィールドの初期化（GC, フォント，widget 特有のデータなど）

形式は

```
typedef void (*XtInitProc)();

void InitProc (request, new)
   Widget  request, new;
```

です。new が新しく作成されるインスタンス・レコードです。request は widget のデフォルトとリソース・データベースと ArgList を元に決められたリソースの生の値を持った作業用のインスタンス・レコードです。これを元に initialize プロシージャは new の widget にリソースを設定します。スーパー・クラスに含まれるリソースはスーパー・クラスの initialize プロシージャによってすでに初期化されています。request と new を比較してスーパー・クラスの initialize プロシージャによって変更されたフィールドをチェックする事もできます。

initialize_hook プロシージャは widget のサブ・リソースを初期化するのに用います。サブ・リソースというのは通常の widget レコードには直接含まれていない（別に管理されている）widget のデータです。例えば Text Widget の source と sink はこの形式になっています。

```
typedef void (*XtArgsProc)()

void ArgsProc (w, args, num_args)
   Widget  w;
   ArgList  args;
   Cardinal  *num_args;
```

args には XtCreateWidget で指定された ArgList が渡されます。

initialize, initialize_hook は共に NULL を指定して省略することができます。また，initialize プロシージャはスーパー・クラスのものから順にサブ・クラスのものが呼ばれます。initialize_hook はそのクラスの initialize のすぐ後に呼ばれます。

サブ・リソースを ArgList から取り出すために次の関数が用意されています。

```
void XtGetSubresources (w, base, name, class, resources,
                                num_resources, args, num_args)
   Widget  w;
   caddr_t  base;
   String  name;
   String  class;
   XtResourceList  resources;
   Cardinal  num_resources;
   ArgList  args;
   Cardinal  num_args;
```

base　　サブ・リソースの値が格納される領域（サブ・パート）のアドレス

name, class　　サブ・パートの名前とクラス名（widget のインスタンス名とクラス名に相当）

resources, num_resources　　サブ・リソースの内容を定義したリソース・リスト

サブ・パートに設定されるサブ・リソースの値は，通常のリソースと同様，args，リソース・データベース，resources のデフォルトを順に参照して決められます（XtResourceList の形式については後述）。

5.2.4. widget のウィンドウの作成（realize）

widget の用いるウィンドウは XtRealizeWidget が呼ばれた時に作られます。これはその widget クラスの realize プロシージャによってなされます。realize プロシージャの形式は

```
typedef void (*XtRealizeProc)();

void realizeProc (w, value_mask, attributes)
   Widget  w;
   XtValueMask  *value_mask;
   XSetWindowAttributes  *attributes;
```

value_mask　　attributes の各フィールドに対応したマスク

となっています。attributes と value_mask のうち，次の属性は Core パートのデータを用いてあらかじめ設定されています。

- background_pixmap と background_pixel
- border_pixmap と border_pixel

- o event_mask。設定されているイベント・ハンドラとトランスレーション・テーブル，expose プロシージャがあるかどうか，visible_interest の値に基づく値
- o bit_gravity。expose プロシージャがない（NULL）場合は NorthWestGravity
- o do_not_propagate_mask。すべてのポインタ・イベントとキーボード・イベントが無効となっている

これ以外の属性は realize プロシージャで設定します。

realize プロシージャに XtInheritRealize を指定することによって Core のものを継承することができます。Core の realize プロシージャは windowClass と visual を CopyFromParent に設定してウィンドウを作ります。

また，widget のウィンドウを作成するために次の関数が用意されています。

```
void XtCreateWindow (w, window_class, visual, value_mask, attributes)
   Widget  w;
   unsigned int  window_class;
   Visual  *visual;
   XtValueMask  value_mask;
   XSetWindowAttributes  *attributes;
```

XtCreateWindow は Core パートの depth, screen, parent->Core.window, x, y, width, height, border_width の値を用いてウィンドウを作ります。作られたウィンドウの ID はインスタンス・レコードの window のフィールドに設定されます。

5.2.5. widget の書き直し（expose, compress_exposure, visible_interest）

クラス・レコードの expose プロシージャによって widget の exposure イベントの処理を行ないます。

```
typedef void (*XtExposeProc)();

void ExposeProc (w, event, region)
   Widget  w;
   XEvent  *event;
   Region  region;
```

クラス・レコードの compress_expose フィールドが True の場合は複数の exposure イベントの領域を合成したものが event に渡されます。クラス・レコードの visible_notify を True に指定するとウィンドウが見えていない間，インスタンス・レコードの visible フィールドが False になります。この情報を用いて無駄なウィンドウの書き直しのために処理を省くことができます。ちなみに intrinsic は VisibilityNotify イベントを用いて visible フィールドの設定を行なっています。

5.2.6. widget のデータの解放（destroy）

XtDestroyWidget で widget が削除される過程で destroy プロシージャが呼ばれます。destroy プロシージャは initialize プロシージャと initialize_hook プロシージャでアロケートしたメモリやリソースを解放するために用います。形式は

```
typedef void (*XtWidgetProc) ()

void destroy (widget)
   Widget  widget;
```

です。destroy プロシージャはサブ・クラスのものから順に呼ばれます。NULL を指定して省略することもできます。

また，destroyCallback というリソースで指定されるコールバック・プロシージャは widget を使用するアプリケーションが widget が削除される時に後処理を行なうためのものです。destroyCallback は CorePart の destroy_callbacks というフィールドで保持されています。CorePart の geing_destroy は widget の削除の処理の間 True となり，widget の削除の処理の重複を避ける働きをしています。

5.2.7. リソースの定義（resources, num_resources）

クラス・レコードの resources と num_resources で widget にどういうリソースがあるのか，それはどういう形式をしているのか，の指定を行ないます。resources は XtResources の配列で，リソース・リストと呼びます。

```
typedef struct _XtResource {
   String    resource_name;   /* Resource name */
   String    resource_class;  /* Resource class */
   String    resource_type;   /* Representation type desired */
   Cardinal  resource_size;   /* Size in bytes of representation */
   Cardinal  resource_offset; /* Offset from base to put resource value */
   String    default_type;    /* representation type of specified default */
   caddr_t   default_addr;    /* Address of default resource */
} XtResource;

typedef struct _XtResource *XtResourceList;
```

resource_name	リソースの名前。リソース名は小文字で始まり，単語間の区切りでは後ろの単語の 1 文字目を大文字にする（"borderWidth", "backgroundPixmap" など）。実際には名前に対応したシンボルを用いる（XtNborderWidth, XtNbackgroundPixmap など）
resource_class	リソースの属するリソース・クラスの名前。リソースクラスの名前は大文字で始める
resource_type	インスタンス・レコード上でのリソースの実体の型の指定。リソース・タイプも大文字で始まる
resource_size	リソースの実体の大きさ（byte 数）
resource_offset	インスタンス・レコード上のリソースを格納するフィールドの先頭からのオフセット
default_type	default_addr で指定する値の型
default_addr	リソースのデフォルト値へのポインタ。値は default_addr の型で指定する

intrinsic はリソース・リストの基づいて ArgList やリソース・ファイルなどの指定を解析し，widget のリソースの設定を行ないます。

スーパー・クラスで定義されているリソースについては再び定義する必要はありません。これを重複して定義することによってデフォルトの値などの変更を行なうこともできます。

intrinsic で定義されているリソース名，リソース・クラス名，リソース・タイプは <X11/StringDefs.h> に宣言されています。このファイルではそれらのシンボル名として，リソース名には XtN, リソース・クラス名には XtC, リソース・タイプには XtR をそれぞれの実際の名前の前につけています（XtNforeground, XtCForeground, XtRPixel など）。ここにない名前を用いる場合は各 widget のパブリック・ヘッダファイルで宣言します。

リソース・クラスは widget で用いるリソースを分類し，それらに対する値の指定をまとめて行なえるようにするものです。例えば，foreground と borderColor を同じ Foreground のクラスにするとか，複数のフォントの指定を Font クラスでまとめたりすることができます。

リソース・タイプは次のものが <X11/StringDefs.h> に定義されています。

XtRAcceleratorTable, XtRBool, XtRBoolean, XtRCallback, XtRColor, XtRCursor, XtRDimension, XtRDisplay, XtREditMode, XtRFile, XtRFont, XtRFontStruct, XtRFunction, XtRGeometry, XtRInt, XtRJustify, XtRLongBoolean, XtROrientation, XtRPixel, XtRPixmap, XtRPointer, XtRPosition, XtRShort, XtRString, XtRStringTable, XtRTranslationTable, XtRUnsignedChar, XtRWindow, XtRCallProc, XtRImmediate

リソース・タイプはリソース・ファイル上のリソースの指定（文字列）や，リソース・リスト上のデフォルトの指定をインスタンス・レコード上に格納する値に変換するのに用います。例えば，XtNforeground の指定はリソース・ファイル上では "White" などの文字列ですが，インスタンス・レコード上ではピクセル値です。この実際の変換の処理はリソース・コンバータ（後述）が行ないます。XtRCallProc（値を求める関数）と XtRImmediate（そのままの値）はリソース・リスト上でだけ用いる特別なリソース・タイプです。

resource_offset を求めるために次のマクロが用意されています。

```
#define XtOffset(type,field)  ((unsigned int)&(((type)NULL)->field))
```

5.2.7.1. リソース・コンバータ

リソース・コンバータはリソースの型変換を行ないます。intrinsic は次に挙げるような Core widget で用いるすべてのリソースに対するコンバータを用意しています。

XtRString から
XtRAcceleratorTable, XtRBool, XtRBoolean, XtRCursor, XtRDimension, XtRDisplay, XtRFile, XtRFloat, XtRFont, XtRFontStruct, XtRInt, XtRPixel, XtRPosition, XtRShort, XtRTranslationTable, XtRUnsignedChar へ

XtRColor から XtRPixel へ

XtRInt から
XtRBool, XtRBoolean, XtRCallback, XtRColor, XtRDimension, XtRFloat, XtRFont, XtRPixel, XtRPixmap, XtRPosition, XtRShort, XtRUnsignedChar へ

XtRPixel から XtRColor へ

widget で作成したリソース・コンバータの登録は次の関数で行ないます。

```
void XtAddConverter (from_type, to_type,
                     converter, convert_args, num_args)
   String  from_type;
   String  to_type;
   XtConverter  converter
   XtConvertArgList  convert_args;
   Cardinal  num_args;
```

convert_args はリソース・コンバータが型変換をする際に必要となる情報を指定するために用います。次のような構造体の配列です（<X11/Convert.h> で定義されている）。

```
typedef enum {
   XtAddress,          /* addreess */
   XtBaseOffset,       /* offset */
   XtImmediate,        /* constant */
   XtResourceString,   /* resource name string */
   XtResourceQuark     /* resource name quark */
} XtAddressMode;

stypedef struct {
   XtAddressMode  address_mode;
   caddr_t  address_id;
   Cardinal  size;
} XtConvertArgRec, *XtConvertArgList;
```

address_mode は address_id で指定するデータがどういうデータなのかを指定します。XtAddress の指定で address_id が静的なデータのアドレス，XtBaseOffset でインスタンス・レコード中のオフセット，XtImmediate で定数，XtResourceString でリソース名，XtResourceQuark でリソース名に対応する quark 値（XrmQuark）となります。size はデータの byte 数です。これらを元に intrinsic はリソース・コンバータに渡す引き数リストを作成します。リソース・コンバータは次の形式で呼ばれます。

```
typedef void (*XtConverter) ();

void Converter (args, num_args, from, to)
   XrmValue  *args;
   Cardinal  *num_args;
   XrmValue  *from;
   XrmValue  *to;
```

XtAddResourceConverter で指定した convert_args に従って args が渡されます。これらを元に from を変換して to に返します。XrmValue は次のような構造体（ディスクリプタ）です。

```
typedef struct {
   unsigned int  size;
   caddr_t  addr;
} XrmValue, *XrmValuePtr;
```

widget のリソース・コンバータを直接呼ぶために次の関数が用意されています。

```
void XtConvert (w, from_type, from, to_type, to_return)
   Widget  w;
   String  from_type;
   XrmValuePtr  from;
   String  to_type;
   XrmValuePtr  to_return;
```

w は変換に必要な他の情報（リソース・コンバータの args）を得るための widget です。

5.2.8. リソースの設定と参照（set_values, set_values_hook, set_values_almost, get_values_hook）

アプリケーション・プログラムは XtSetValues と XtGetValues を用いて widget のリソースを設定，参照します。

XtSetValues は widget の set_values プロシージャをスーパー・クラスからサブ・クラスの順で呼び出します。set_values_hook プロシージャがあれば（NULL でない）各クラスの set_values プロシージャのすぐ後に呼びます。set_values_hook プロシージャはサブ・パートのリソースを設定するためのものです。

そしてもし，widget のジオメトリの変更が要求されていると，XtSetValues はジオメトリ・マネージャにそのことを知らせます。そしてジオメトリ・マネージャが XtGeometryYes を返せば XtSetValues は widget の resize プロシージャを呼びます。XtGeometryNo（ジオメトリの変更が許されない）ならば，widget のジオメトリを元にもどします。XtGeometryAlmost（ある程度の変更を認める）ならば，set_values_almost プロシージャを呼びます。

XtSetValues は最後に，set_values プロシージャの戻り値をチェックします。True のものがあれば，widget の window を書き直す（消した後 expose イベントを起こす）ために Xlib の XClearArea を呼びます。

set_values プロシージャは次の形式で作成します。

```
typedef Boolean (*XtSetValuesFunc) ();

Boolean SetValuesFuc (current, request, new)
   Widget  current;
   Widget  request;
   Widget  new;
```

current が widget のもとの状態，request が XtSetValues で要求された状態，new が目的とするインスタンスレコードです。set_values プロシージャは initialize プロシージャと同様，ポインタで指定されるリソースの領域の確保，コピーとリソースによって決まる他の値の設定などを行ないます。その結果，widget を書き直す必要がある場合には戻り値として True を返します。これによって，widget の消去と expose プロシージャの呼び出しが自動的に行なわれます。set_values プロシージャは NULL を指定して

省略することができます。

set_values_hook プロシージャは widget のサブ・パートのリソースを設定するためのプロシージャです。

```
typedef Boolean (*XtArgFunc) ();

Boolean ArgFunc (w, args, num_args)
   Widget  w;
   ArgList  args;
   Cardinal  *num_args;
```

サブ・パートのリソースを設定するために次の関数が用意されています。

```
void XtSetSubcalues (base, resources, num_resources,
                              args, num_args)
   caddr_t  basel
   XtResourceList  resources;
   Cardinal  num_resources;
   ArgList  args;
   Cardinal  num_args;
```

set_values_hook プロシージャは NULL を指定して省略することができます。

set_values_almost プロシージャはジオメトリ・マネージャの XtGeometryAlmost の対応するためのプロシージャです。形式は

```
typedef void (*XtAlmostProc) ();

void AlmostProc (w, new_widget_return,
                           request, reply)
   Widget  w;
   Widget  new_widget_return;
   XtWidgetGeometry  *request;
   XtWidgetGeometry  *reply;
```

です。w がもとの XtGeometryAlmost を出す原因となった widget, request は w に基づいたジオメトリ指定, reply はジオメトリ・マネージャによって示されたジオメトリ（の折衷案）です。これらを元に new_widget_return に新しくジオメトリを設定して返します。set_values_almost プロシージャに XtInheritSetValuesAlmost を指定して, Core の用意するプロシージャを使用することもできます。このプロシージャはジオメトリ・マネージャの折衷案をそのまま受け入れます。

XtGetValues が呼ばれると intrinsic はリソース・リストの情報を元にインスタンス・レコードからリソースの値を得ていきます。通常のリソースはこれで参照できます。サブ・パートで管理されているサブ・リソースについては widget の get_values_hook プロシージャによって参照されます。これは

```
typedef void (XtArgsProc) ();

void ArgsProc (eidget, args, num_args)
   Widget  widget;
   ArgList  args;
   Cardinal  num_args;
```

と定義します。widget にサブ・パートがない場合は get_values_hook に NULL を指定します。サブ・リソースを参照するために次の関数が用意されています。

```
void XtGetSubValues (base, resource, num_resources,
                        args, num_args)
   caddr_t  base;
   XtResourceList  resources;
   Cardinal  num_resources;
   ArgList  args;
   Cardinal  num_args;
```

args と num_args には intrinsic から渡された args と num_args をそのまま指定することができます。

5.2.9. アクション・リスト (actions, num_actions, tm_table)

インスタンスのトランスレーション・テーブルで指定されるアクションは次の順序で検索されます。

i. その widget の widget クラスのアクション・テーブル

ii. その widget のスーパー・クラス（およびその先祖のクラス）のアクションテーブル

iii. XtAddActions で定義されたアクション

widget クラスのインスタンスが使用するアクションはクラス・レコードの actions (XtActionList) と num_actions で設定します。XtActionList は次のように定義されています。

```
typedef struct _XtActions {
   String  action_name;
   XtAcrionProc  action_proc;
} XtActionRec, *XtActionList;
```

action_name がアクション・テーブルで用いるアクション名，action_proc がアクションを処理する関数です。これは，

```
void ActionProc (w, event, params, num_params)
   Widget  w;
   XEvent  *event;
   String  *params;
   Cardinal  *num_params;
```

という形式呼び出されます。アクション・テーブルで指定した引き数が params として文字列で渡されます。

widget のデフォルトのトランスレーション・テーブルは tm_table に String で指定します。これは intrinsic のトランスレーション・マネージャによって自動的に解析され，インスタンス・レコードの tm に設定されます。また，スーパークラスのトランスレーション・テーブルをそのまま引き継ぐには XtInheritTranslations を tm_table に指定します。

5.2.9.1. イベント・ハンドラ

通常，widget のイベントと処理の対応付けは前述のトランスレーション・テーブルで行ないます。これ以外の方法として，イベント・ハンドラによる方法があります。イベント・ハンドラは個々のイベントと直接対応付けて設定されます。

イベント・ハンドラの設定，解除は次の関数で行ないます。

```
void XtAddEventHandler (w, event_mask, nonmaskable,
                        proc, client_data)
   Widget  w;
   EventMask  event_mask;
   Boolean  nonmaskable;
   XtEventHandler  proc;
   caddr_t  client_data;

void XtRemoveEventHandler (w, event_mask, nonmaskable,
                           proc, client_data)
   Widget  w;
   EventMask  event_mask;
   Boolean  nonmaskable;
   XtEventHandler  proc;
   caddr_t  client_data;
```

nonmaskable はイベント・マスクで指定できないイベント（GraphicsExpose, NoExpose, SelectionClear, SelectionRequest, SelectionNotify, ClientMessage, MappingNotify）が起こった時にイベント・ハンドラを起動するかどうかの指定です。これと event_mask で指定したイベントが起こるとイベント・ハンドラが起動されます。client_data はそのままイベント・ハンドラに渡されます。イベント・ハンドラは次の形式で呼ばれます。

```
typedef void (*XEventHandler) ();

void EventHandler (w, client_data, event)
   Widget  w;
   caddr_t  client_data;
   XEvent  *event;
```

イベント・ハンドラによるイベントと処理の対応付けはアプリケーション・プログラムで行なうこともできます

5.2.10. その他のフィールド

Core クラス・パートのその他のフィールドは次の通り。

superclass　スーパー・クラスのクラス・レコードを指定する

class_name　widget クラス名を指定

widget_size　インスタンス・レコードの大きさ（sizeof (widgetRec)）を指定

xrm_class　リソース・マネージャが使用

compress_motion
　イベント・キューにたまっている MotionNotify イベントを圧縮するかどうか

compress_enterleve
　イベント・キューにたまっている EnterNotify と LeveNotify を圧縮するかどうか

resize　widget がリサイズされた時に呼ばれるプロシージャ（XtWidget-Proc）。リサイズにともなって処理を行ないたい場合に指定

accept_focus　インプット・フォーカスを受け付けるかどうかと，インプット・フォーカスされた時に呼ばれるプロシージャ。NULL を指定すると受け付けない

version　X Toolkit のバージョンの指定。XtVersion を指定する

callback_private
　intrinsic がコールバック・リストを操作する使用

query_geometry
　サブ widget のためのジオメトリ・ハンドラの指定

display_accelerator
　アクセラレータを表示するためのプロシージャ（XtStringProc）

widget の定義　　widget の内部構造とその定義

extension　　将来のためのフィールド

Chapter 6

Athena widget

Athena widget は MIT の Athena プロジェクトでサンプルとして開発された X Toolkit 上の最初の widget ライブラリです。Athena widget は多くの X のインプリメンテーションで利用することができます（ただし，Athena widget は X Consortium の定める X standard には含まれていません）。

6.1. Athena widget の widget

Athena widget には次の widget が含まれています。

名前（クラス名）	機能
コマンド・ボタン（Command）	ラベルのついたボタンをクリックすることによってあらかじめ指定したコールバック・プロシージャを起動する
ラベル（Label）	ウィンドウに文字列（ラベル）を表示する
テキスト（Text）	テキストの表示とキーボードとマウスによるエディットを行なうための widget。メモリ中の文字列とテキスト・ファイルの両方を扱うことができる
スクロールバー（Scrollbar）	縦方向や横方向のスクロールバーを表示し，マウス操作によってスクロールを行なうための基本 widget
ビューポート（Viewport）	スクロールバーを用いてサブ widget を自動的にスクロールして表示させる widget
ボックス（Box）	サブ widget を縦や横にならべて矩形の中に配置する
VPaned（VPaned）	複数のサブ widget を縦方向にならべ，その境界をマウスで自由に移動してサブ widget の大きさを変更することができる widget
フォーム（Form）	複数の widget を指定した位置関係に従って配置する widget
ダイアログ（Dialog）	ラベル，テキスト入力エリア，複数のボタンを持った widget。ユーザにプロンプトを出し，入力や確認を要求するために用いる
グリップ（Grip）	ウィンドウ上の小さな領域でポインタ・イベントを受け付けるために用いる widget

Athena widget	Athena widget の widget
リスト（List）	複数の文字列を縦横にならべ，マウス操作によってコールバック・プロシージャを起動する widget

6.2. Athena widget を用いたプログラミング

Athena widget は基本的な widget しか用意していませんが，その分利用するのは簡単です。ただし，アプリケーションから直接グラフィックを出力するための widget がありません。このため，Athena widget の内部で使われている Sample widget にイベント・ハンドラを定義して出力するとか，簡単な専用の widget を定義する，などの工夫が必要です。

Athena widget を用いたプログラムでは XToolkit intrinsic のヘッダ・ファイルのインクルードのために

```
#include <X11/Intrinsic.h>
#include <X11/Stringdefs.h>
```

と，それぞれの widget のヘッダ・ファイル（<X11/Label.h>, <X11/Command.h> など）のインクルードを行ないます。コンパイル時には

```
cc file.c -lXaw -lXmu -lXt -lX11
```

と指定します（VAXstation と DECstation には Xmu のライブラリがシステムに含まれていないので，これを自分で用意する必要があります。また，X Toolkit のヘッダファイルが /usr/include/mit/X11 にあるのでコンパイル時に -I/usr/include/mit の指定も必要です）。

Athena Widget に限らず，X Toolkit を用いたプログラムはリソース・ファイルや引数の指定によって widget のリソースを起動時に変更することもできます。

例として Command Widget を用いた "Hello World!" プログラムを示します。マウスの操作によって起動される処理がコールバック・プロシージャの形で書かれています。

```
/*
 * hw_xaw.c
 *      sample implementation of hw on Xaw
 *      to compile this (on ULTRIX), enter;
 *              % cc hw_xaw.c -o hw_xaw -lXaw -lXmu -lXt -lX11
 *                      -I/usr/include/mit -I/usr/local/include
 *      (in this case, libXmu.a is located on /usr/local/lib/ and
 *       Xmu.h is on /usr/local/include/X11/)
 *
 * Tat001       18-Jul-1989     for x11pg
 */

#include <X11/Intrinsic.h>
#include <X11/StringDefs.h>

#include <X11/Command.h>

static char     *hello_world = {"Hello world!"};
void hw_callback();

main (argc, argv)
int     argc;
char    *argv[];
{
        Widget  toplevel, hw;
        Arg     argl[4];
        int     i;
        static XtCallbackRec hw_callbacks[] = {{hw_callback, 0}, {0, 0}};

        /* initialize the toolkit and create a top-level shell */
        toplevel = XtInitialize ("hw on Xaw", "Test", 0, 0, &argc, argv);

        /* create a command button */
        i = 0;
        XtSetArg (argl[i], XtNlabel, hello_world); i++;
        XtSetArg (argl[i], XtNcallback, hw_callbacks); i++;
        hw = XtCreateManagedWidget ("hw", commandWidgetClass, toplevel,
                        argl, i);

        XtRealizeWidget (toplevel);

        XtMainLoop ();
}

/* callback procedure for command button */
void hw_callback (widget, user_data, call_data)
Widget  widget;
int     user_data, call_data;
{
```

```
        exit ();
}

/* end of hw_xaw.c */
```

Chapter 7

XUI Toolkit

7.1. XUI 概要

XUI (X User Interface) は DECwindows 上で動作するアプリケーションの標準的なグラフィカル・ユーザ・インターフェースです。

XUI のユーザ・インターフェースを実現するために X Toolkit の上位に XUI Toolkit が構築されています。XUI Toolkit は Push Button, Menu, File Selection, Help など数多くの実用的な widget を提供しています。また, XUI Toolkit は単に widget を提供するだけではなく, UIL と DRM によって X Toolkit の機能をさらに拡張しています。

UIL (XUI User Interface Language) はアプリケーションのユーザ・インターフェースを記述するための一種の簡易言語です。UIL を用いることによってアプリケーションの本来の処理とユーザインターフェースのための処理を切り離すことができます。インターフェースの変更やカスタマイズも UIL を変更するだけで容易に行なえます。

DRM (XUI Resource Manager) は UIL の定義を元にして widget の作成, 表示, 操作を実行時に行ないます。コールバック・プロシージャの設定も DRM によって自動的になされます。

XUI は X Toolkit では規定していなかったユーザ・インターフェースのスタイルを XUI Style Guide という形で定義しています。XUI Style Guide は widget の見た目や形状 (appearance), マウス・ボタンの使い方と widget の動き (beavior) を決め, アプリケーション内, アプリケーション間でのユーザ・インターフェースの統一を計っています。XUI Toolkit は XUI Style Guide に沿う形で作られています。

7.2. XUI Toolkit widget 概要

XUI Toolkit には XUI Style Guide に示されたユーザ・インターフェースを実現するための数多くの widget が用意されています。これらの widget はたがいに関連し合ってアプリケーションを構成します。

XUI Toolkit では X Toolkit intrinsic の関数とは別に, 個々の widget に対して低レベルの widget 作成関数と高レベルの widget 作成関数を用意しています。低レベルの widet 作成関数は intrinsic の XtCreateWidget と同様, ArgList を用いて widget の作成を行ないます。リソースを細かく指定することによって, アプリケーション独自のスタイルを実現することもできます。高レベルの widget 作成関数はおもなリソースを引き数で直接受け取って widget を作成します。高レベルの widget 作成関数は ArgList を用いない分だけシンプルです。これとは別に, UIL によって widget を指定することもできます。UIL を用いると一連の widget が DRM によって自動的に作成されます。

また, XUI Toolkit では gadget というタイプの widget も用意しています。これらは

一般の widget に比べて簡単な構造になっています（widget クラスの階層が浅く，スーパー・クラスが低レベル。その上ウィンドウも作らない）。このため gadget はメモリ，速度共に同じ widget より効率が良くなっています。Label, Separator, Push Button, Toggle Button の 4 種類の gadget があります。

XUI Toolkit の提供する widget を以下に示します。かっこ内は各 widget の UIL でのオブジェクト名，低レベルの widget 作成関数，高レベルの widget 作成関数（および gadget 作成関数）の関数名です。

o ウィンドウ widget
アプリケーションのトップ・レベル・ウィンドウに直接関連する widget 群

Main Window タイトルバーを持ったアプリケーションのトップ・レベル・ウィンドウを構成する widget。メニューバーとコマンド・ウィンドウ，スクロール・バーを直接管理する（main_window, DwtMainWindow, DwtMainWindowCreate）

Menu Bar プルダウン・メニューのメニュー・エントリを管理，配置するための widget （menu_bar, DwtMenuBar, DwtMenuBarCreate）

Window アプリケーションが直接描画を行なうためのウィンドウ（window, DwtWindow, DwtWindowCreate）

Scroll Window スクロール・バーの操作によって自動的にサブ widget をスクロールさせる widget （scroll_window, DwtScrollWindow, DwtScrollWindowCreate）

o サブ・エリア widget
アプリケーション・ウィンドウのワーク・エリアを構成するのに用いる，スクロール・バー widget とコントロール・パネルのための widget 群

Scroll Bar ウィンドウに表示されている部分がデータ全体に対してどの辺りの位置でどれだけの割合を占めているのかを表示する widget （scroll_bar, DwtScrollBar, DwtScrollBarCreate）

Label テキストやアイコンを固定的に表示する widget （label, DwtLabel, DwtLabelCreate, DwtLabelGagetCreate）

Toggle Button on か off かの二値の値をユーザに指定させるラベルの付いたボタンまたはアイコン（toggle_button, DwtToggleButton, DwtToggleButtonGadgetCreate）

Radio Box 複数のボタンの中からひとつだけを選ばせる widget。ラジオの選局ボタンが名前の由来（radio_box, DwtRadioBox,

DwtRadioBoxCreate)

Push Button　選択することによって，あらかじめ設定した動作が起動するボタン（push_button, DwtPushButton, DwtPushButton-Create, DwtPushButtonGadgetCreate)

Scale　ある決まった範囲の中の任意の値をユーザに指定させるための widget （scale, DwtScale, DwtScaleCreate)

o メニュー widget
メニューを作るための widget 群

Pull-Down Menu　プルダウン・メニューの本体となる widget （pulldown_menu, DwtMenu, DwtMenuPulldownCreate)

Pop-Up Menu　ユーザが MB2 を押した時にポインタの位置に表示されるタイプのメニュー（popup_menu, DwtMenu, DwtMenuPopup-Create)

Work Area Menu　ワークエリアに常に表示されるメニュー。単にボタンをならべるためのもの（work_area_menu, DwtMenu, DwtMenu-Create)

Pull-Down Menu Entry
プルダウン・メニューを出現させるためのボタン。メニュー・バー上で用いる（pulldown_entry, DwtPullDown-MenuEntry, DwtPullDownMenuEntryCreate)

Option Menu　ある項目に対する値を，複数の選択肢の中から選ばせるメニュー。デフォルトの値が最初に表示される （option_menu, DwtOptionMenu, DwtOptionMenuCreate)

Separator　リストやメニューの項目を区切るために用いる点線のラベル（separator, DwtSeparator, DwtSeparatorCreate, DwtSeparatorGadgetCreate)

o ダイアログ・ボックス widget
ダイアログ・ボックスは，プルダウン・メニューでは指定しきれないまとまった量のデータをユーザに指定させたり，メッセージを表示し返答を得るなど，ユーザと情報を直接やり取りするのに使われる特別なウィンドウ。ダイアログ・ボックスを作る際にその基本となる widget 群

Dialog Box　最も基本的なダイアログ・ボックス。表示すべきラベルやボタンをこの widget のサブwidget として作成する。ダイアログ・ボックスはウィンドウ・マネージャから直接制御される （dialog_box, DwtDialogBox, DwtDialogBoxCreate)

Pop-up Dialog Box

ペアレント widget によってクリッピングされないダイアログ・ボックス（popup_dialog_box, DwtDialogBox, DwtDialogBoxPupupCreate）

Attached Dialog Box
サブ widget の配置をボーダーや他の widget との相対的な位置で指定できるダイアログ・ボックス。リサイズされると自動的に再配置が行なわれる（attached_dialog_box, DwtAttachedDB, DwtAttachedDBCreate）

Pop-up Attached Dialog Box
ペアレント widget によってクリッピングされない Attached Dialog Box （popup_attached_db, DwtAttachedDB, DwtAttachedDBPopupCreate）

o テキスト widgetテキストの入出力をするための widget と項目を選択するための widget

Simple Text テキストの入力，出力，エディットをするための widget。簡易的なテキスト・エディタとしても利用できる（simple_text, DwtSText, DwtSTextCreate）

List Box たくさんの項目からなるリストを表示し，その項目を選択させる widget。スクロール・バーによってユーザが表示範囲を移動することができる（list_box, DwtListBox, DwtListBoxCreate）

o 標準ダイアログ・ボックス
XUI で定義されている標準的なダイアログ・ボックスを作るための widget

Help ヘルプ・ファイルからヘルプ・ウィンドウを構成する widget （help_box, DwtHelp, DwtHelpCreate）

Work-in-Progress Box
時間のかかる処理の最中であることをユーザに示すためのダイアログ・ボックス（work_in_progress_box, DwtWorkBox, DwtWorkBoxCreate）

Message box ユーザに対して一般のメッセージを告げるためのダイアログ・ボックス（message_box, DwtMessageBox, DwtMessageBoxCreate）

Caution Box 警告メッセージを出力するためのダイアログ・ボックス。ファイルを消すなど逆戻りできない操作の前にユーザに確認を促すために用いる（caution_box, DwtCautionBox, DwtCautionBoxCreate）

Command Window	コマンド入力エリアのための widget。コマンドをキーボードから入力するための標準的な形式（command_window, DwtCommandWindow, DwtCommandWindowCreate）
Selection Box	リストの中から項目のひとつをユーザに選択させるダイアログ・ボックス（selection, DwtSelection, DwtSelectionCreate）
File Selection	ユーザにファイルの指定をさせるためのダイアログ・ボックス（file_selection, DwtFileSelection, DwtFileSelectionCreate）

7.2.1. widget 作成関数とコールバック

XUI Toolkit は各 widget に対して低レベルと高レベルの 2 つの widget 作成関数を提供しています。例えば Label widget でこれらは次のようになっています。

```
Widget DwtLabel (parent_widget, name, x, y, label, help_callback)
   Widget  parent_widget;
   char  *name;
   Position  x, y;
   DwtCompString  label;
   DwtCallbackPtr  help_callback;

Widget DwtLabelCreate (parent_widget, name,
                       override_arglist, override_argount)
   Widget  parent_widget;
   char  *name;
   ArgList  override_arglist;
   int  override_argount;
```

Dwt<widget> が高レベル，Dwt<widget>Create が低レベルの作成関数です。高レベルの作成関数はそれぞれの widget で頻繁に使うリソースが引き数で直接指定できるようになっています。他のリソースについては XUI Style Guide に合う形のデフォルト値を持っています。低レベルの作成関数ではリソースを ArgList の形で設定します。引き数の形式はどの widget でも同じです。

コールバック・リスト（DwtCallbackPtr）は X Toolkit intrinsic と同じ形式です。

```
typedef struct {
   VoidProc  proc;
   int  tag;
} DwtCallback, *DwtCallbackPtr;
```

配列の最後は NULL のデータでターミネイトしておきます。コールバック・プロシージャ（proc）は次の形式で呼び出されます。

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

void porc (widget, tag, callback_data)
   Widget  widget;
   Opaque  tag;
   DwtAnyCallbackStrunt  callback_data;
```

tag に コールバック・リストで指定したデータがそのまま渡されます。DwtAnyCallback-Struct は汎用的な構造体で widget 毎に違っています。このうち，reason はどういう種類のコールバックなのかを示します。値としては DwtCRActivate, DwtCRExpose, DwtCRValueChanged などが widget 毎に用意されています。event はそのコールバックの原因となったイベントです。

このように XUI Toolkit ではコールバックの種類が reason という形で整理されているので，複数のコールバックを 1 つのプロシージャで共有することが楽にできます。

UIL ではコールバック・リストは次のように指定します。

```
CALLBACKS {
   reason = PROCEDURE proc (tag);
};
```

7.3. ウィンドウ widget

X Toolkit のアプリケーションは XtInitialize が作る toplevel widget によってその外形が作られます。XUI Style Guide ではこの中に上から順に，タイトル・バー，メニュー・バー，ワーク・ウィンドウ（ワーク・エリア）を置くように定めています。ワーク・ウィンドウには縦と横のスクロール・バーをつけることもできます。これらのための widget として Main Window, Menu Bar, Window, Scroll Window が用意されています。タイトル・バーはウィンドウ・マネージャが作ります。

Main Window widget はメニュー・バー，ワーク・ウィンドウ，スクロール・バー，コマンド・ウィンドウ（キーボードによるコマンド入力を行なうためのウィンドウ，ワーク・ウィンドウの下に置かれる）を並べるための widget です。Main Window Widget は通常，toplevel widget のすぐ下のサブ widget として作成します。widget 作成関数として

```
DwtMainWindow (parent_widget, name, x, y, width, height)
DwtMainWindowCreate (parent_widget, name,
                     override_arglist, override_argcount)
```

が用意されています。また，サブ widget を登録し，並べるための関数

```
DwtMainSetAreas (widget, menu_bar, work_window, command_window,
                 horizontal_scroll_bar, vertical_scroll_bar)
```

があります。これらはリソース（DwtNmenuBar など）で指定することもできます。サブ widget の設定は Main Window widget を realize する前に行なう必要があります。コマンド・ウィンドウはワーク・ウィンドウの下に置かれます。

Menu Bar widget はメニュー・エントリを並べるための widget です。ヘルプのためのエントリは右端に置かれます。スーパー・クラスは Menu widget です。widget 作成関数は

```
DwtMenuBar (parent_widget, name, entry_callback, help_callback)
DwtMenuBarCreate (parent_widget, name,
                  override_arglist, override_argcount)
```

です。entry_callback（リソース DwtNentryCallback）はサブ widget の activate コールバック（サブ widget の DwtNactivateCallback）に置き変わるコールバックの指定です（通常はそれぞれのメニュー・エントリのコールバックを使用する）。このコールバック構造体は

```
typedef struct {
   int  reason;
   XEvent  *event;
   Widget  s_widget;
   char  *s_tab;
   char  *s_callbackstruct;
} DwtMenuCallbackStruct;
```

です。

Menu Bar widget のサブ widget として作成した widget は自動的にMenu Bar 上に表示されます。ヘルプ・エントリはサブ widget として作成した後，DwtNmenuHelpWidget というリソースに定義します。また，DwtNspacing でエントリ間の間隔を広げることもできます。

Window widget はアプリケーションのグラフィックを表示するための widget です。この widget を用いると，アプリケーションで独自の widget を定義する必要がほとんどなくなります。widget 作成関数は

```
DwtWindow (parent_widget, name, x, y, width, height, callback)
DwtWindowCreate (parent_widget, name,
                 override_arglist, override_argcount)
```

です。callback は widget の内容の書き直しのためのコールバックです（リソース名 DwtNexposeCallback）。コールバック reason は DwtCRExpose，コールバック構造体は

```
typedef struct {
   int  reason;
   XExposeEvent  *event;
   Window  w;
} DwtWindowCallbackStruct;
```

です。

Scroll Window widget はスクロール・バーとワーク・ウィンドウをサブ widget として持ち，スクロール・バーのスライダの大きさをワーク・ウィンドウの大きさに従って自動的に設定します。ただし，ワーク・ウィンドウやスライダの位置の設定やスクロール・バー上のユーザの操作への対応（実際のスクロール）は一切行ないません。作成関数は

```
DwtScrollWindow (parent_widget, name, x, y, width, height)
DwtScrollWindowCreate (parent_widget, name,
                       override_arglist, override_argcount)
```

です。また，

```
DwtScrollWindowSetAreas (widget, horizontal_scroll_bar,
                                 vertical_scroll_bar, work_region)
```

もしくは DwtNhorizontalScrollBar, DwtNverticalScrollBar, DwtNworkWindow のリソースで各サブ widget を設定します。

7.4. サブ・エリア widget

アプリケーション・ウィンドウのワーク・エリアには色々なインターフェース・オブジェクトが置かれます。ここではワーク・エリアで用いる種々の widget を述べます。

Scroll Bar Widget は単体のスクロール・バーです。widget 作成関数は

```
DwtScrollBar (parent_widget, name, x, y, width, height,
              inc, page_inc,
              shown, value, min_value, max_value,
              orientation, value_changed_callback, help_callback,
              unit_inc_callback, unit_dec_callback,
              page_inc_callback, page_dec_callback,
              to_top_callback, to_bottom_callback,
              drag_callback)
DwtScrollBarCreate (parent_widget, name,
                    override_arglist, override_argcount)
```

です。リソース（および対応する引き数）の意味は次の通り（コールバック以外の型はすべて int）。

DwtNinc stepping arrow を操作した時の位置の変化分。ワーク・エリアの一単位分（1 行分など）に相当する値を指定する

DwtNpageIncscroll region で MB1 を押した時の位置の変化分。1 ページ分に相当する値を指定する

DwtNshown ワーク・エリアに表示されている部分の大きさに相当する値を指定する。slider の長さに反映される

DwtNvalue ワーク・エリアに表示されいる部分の先頭の位置に相当する値を指定する。slider の位置に反映される

DwtNminValue, DwtNmaxValue
shown と value のとる値の範囲の指定。テキストの行数やグラフィックのピクセル数など，アプリケーションで都合の良い値を指定すると良い

DwtNorientation
縦（DwtOrientationVertical）か横（DwtOrientationHorizontal）の指定

DwtNvalueCallback
value の値が変化した時に呼ばれるコールバック

DwtNunitIncCallback, DwtNunitDecCallback
stepping arrow が押された時に呼ばれるコールバック

DwtNpageIncCallback, DwtNpageDecCallback
scroll region で MB1 が押された時の呼ばれるコールバック

DwtNtoTopCallback, DwtNtoBottomCallback
scroll reagion で MB2 または MB3 が押された時の呼ばれるコールバック

DwtNdragCallback
slider が（MB1 で）ドラッグされた時に呼ばれるコールバック

コールバック構造体は

```
typedef struct {
   int  reason;
   XEvent  *event;
   int  value;
   int  pixel;
} DwtScrollBarCallbackStruct;
```

となっています。value に value の値，pixel には DwtCRToTop と DwtCRToBottom の際，ボタンが押された位置の scroll region の先頭から測った値が入ります。また，DwtNvalue, DwtNshown などの値を操作するための関数

```
DwtScrollBarGetSlider (widget, value_return, shown_return,
                       inc_return, pageinc_return)
DwtScrollBarSetSlider (widget, value, shown,
                       inc, page_inc, notify)
```

があります。ここで notify は value の指定が有効かどうかの指定（Boolean）です。

Label widget はテキストやアイコンをラベルとして表示するための widget です。Label gadget（テキスト専用）もあります。また，Label widget は Push Button や Toggle Button や Sparator のスーパー・クラスなので，それらの widget でも Label widget の機能を用いることができます。

表示するものがテキストの場合（DwtNlabelType が DwtCString），ラベル（DwtNlabel，UIL では label_label）にコンパウンド・ストリングを指定します。コンパウンド・ストリングには複数のフォントを用いたり，改行（'\n'）を入れたりすることができます。フォント・リスト（DwtFontList）の設定は DwtNfont というリソースで行ないます。アイコンの場合（DwtNlabelType が DwtPixmap）は，ラベル（DwtNpixmap，UIL では label_pixmap）にビックスマップを指定します。

widget 作成関数は

```
DwtLabel (parent_widget, name, x, y, label, halp_callback)
DwtLabelCreate (parent_widget, name,
                override_arglist, override_argcount)
DwtLabelGadgetCreate (parent_widget, name,
                      override_arglist, override_argcount)
```

です。DwtLabel（高レベル関数）はテキストのラベルを作成する場合に用いることができます。

Toggle Button widget は on，off の状態を入力するための widget です。MB1 をクリックすると状態が変化します。この状態は小さな四角い箱またはアイコンで表示されます。スーパー・クラスは Label widget です。widget 作成関数は

```
DwtToggleButton (parent_widget, name, x, y, label, value,
                 value_changed_callback, help_callback)
DwtToggleButtonCreate (parent_widget, name,
                       override_arglist, override_argcount)
DwtToggleButtonGadgetCreate (parent_widget, name,
                             override_arglist, override_argcount)
```

です。value は widget 作成時の on，off を決める Boolean の値です（DwtNvalue）。value_changed_callback は DwtNvalueChangedCallback でユーザの操作によって値が変化した時に呼ばれます。この他のコールバックとしては DwtNarmCallback（MB1 を押した時），DwtNdisarmCallback（MB1 を離した時）があり，それぞれの操作で表示を変化させることもできます。コールバック構造体は

```
typedef struct {
   int  reason;
   XEvent  *event;
   int  value;
} DwtTogglebuttonCallbackStruct;
```

DwtNshape（デフォルト DwtRectangular）は Toggle Button を Radio Box の中で用いる際に，状態表示の箱の形を丸（DwtOval）に変えるために用いられます。アイコンの場合は DwtNpixmapOn と DwtNpixmapOff のリソースにそれぞれのピックスマップを指定します。

value の値を操作するために次の関数が用意されています。

```
DwtToggleButtonGetState (widget)
DwtToggleButtonSetState (widget value, notify)
```

Radio Box widget は複数のトグル・ボタンを並べ，その内からひとつだけを選ばせるための Menu widget です。widget 作成関数は

```
DwtRadioBox (parent_widget, name, x, y, entry_callback, help_callback)
DwtRadioBoxCreate (parent_widget, name,
                        override_arglist, override_argcount)
```

です。entry_callback（DwtNentryCallback）はサブ widget の DwtNvalueChangedCallback に置き換えるコールバックです。コールバック構造体は

```
typedef struct {
   int  reason;
   XEvent  *event;
   Widget  s_widget;
   char   *s_tag;
   char   *s_callbackstruct;
} DwtRadioBoxCallbackStruct;
```

です。s_widget は起動した widget の ID，s_tag はその widget のコールバック・リストで指定した tag，s_callbackstruct はおのおののコールバック構造体です。その他のリソースについてはスーパー・クラスの Menu widget を参照して下さい。

Push Button widget は枠付きのラベルまたはアイコンです。枠の中で MB1 をクリックすると決められた処理が起動します。Push Button widget は通常のボタンの他，プルダウン・メニューの項目など多くの所で用いられてます。スーパー・クラスは Label widget です。widget 作成関数は

```
DwtPushButton (parent_widget, name, x, y, label,
               activate_callback, help_callback)
DwtPushButtonCreate (parent_widget, name,
                     override_arglist, override_argcount)
DwtPushButtonGadgetCreate (parent_widget, name,
                           override_arglist, override_argcount)
```

です。activate_callback はリソース DwtNactivateCallback で MB1 をクリックすると呼ばれます。その他のリソースとしては次のものがあります。

DwtNbordHighlight	枠を二重にするかどうかの指定。Dialog Box のデフォルト・ボタンはこの状態
DwtNfillHighlight	枠の中を塗りつぶすかどうか。MB1 が押された時の状態
DwtNshadow	枠の右下に影をつけるかどうか
DwtNactivateCallback	ボタンが起動（MB1 をクリック）した時に呼ばれるコールバック
DwtNarmCallback	MB1 を押した時に呼ばれるコールバック
DwtNdisarmCallback	MB1 を離した時に呼ばれるコールバック

Scale widget は数直線を使って数値を選ばせる widget です。widget 作成関数は

```
DwtScale (parent_widget, name, x, y, width, height,
          scale_width, scale_height, title,
          min_value, max_value, decimal_points, value,
          orientation,
          value_changed_callback, drag_callback, help_callback)
DwtScaleCreate (parent_widget, name,
          override_arglist, override_argcount)
```

です。各リソースは次の通り。

DwtNscaleWidth, DwtNscaleHeight
: スケール単体の大きさ。ラベルを含めた widget 全体の大きさ（DwtNwidth, DwtNheight）は自動的に計算される

DwtNtitle
: スケールに付けるタイトル（DwtCompString）

DwtNminValue, DwtNmaxValue
: スケールの値の最大値と最小値。デフォルトは 0 から 100

DwtNdecimalPoints

スケール値の表示で小数点をシフトする桁数。デフォルトは 0

DwtNvalue スケールの示す値

DwtNorientation
スケールの縦横の指定。デフォルトは DwtOrientationHorizontal

DwtNshowValue
スライダの位置に値を表示するかどうかの指定。デフォルト True

DwtNvalueChangedCallback
ユーザが値を変化させたときに呼ばれるコールバック

DwtNdragCallback
ユーザがスライダをドラッグしたときに呼ばれるコールバック

スケールの値の表示は DwtNvalue の値の小数点を左に DwtNdecimalPoints だけずらしたもので行われます。例えば DwtNminValue が 0, DwtNmaxValue が 1000 で DwtNdecimal-Value が 2 の時, DwtNvalue は 0 から 1000 の値をとり, 表示は 0.00 から 10.00 になリます。スケールの目盛として用いるラベルはサブ widget として, 小さい値に対応するものから順に作成, 登録します。登録したラベルは等分に配置されます。上の例では, "0.00", "5.00", "10.00" の 3 つのラベルを登録することができます。

DwtNvalue の値を操作するために次の関数が用意されています。

```
DwtScaleGetSlider (widget, value_return)
DwtScaleSetSlider (widget, value)
```

7.5. メニュー widget

複数のトグル・ボタンやプッシュ・ボタンを並べ, ユーザに項目や処理を選ばせるようにしたものがメニューです。XUI Style Guide は次のような形式のメニューを定義しています。

プルダウン・メニュー
メニュー・バーのエントリ上で MB1 を押すことによって（その下にぶら下がるような格好で）現れるメニュー

ポップアップ・メニュー
ワーク・エリア中で MB2 を押すことによって（その位置に）現れるメニュー

ワークエリア・メニュー（コントロール・パネル）
ワーク・エリアに常に表示されているメニュー

サブ・メニュー

プルダウン・メニューやポップアップメニューの項目を選ぶことによって，その中から現れるメニュー

オプション・メニュー
プッシュ・ボタンの上に現れる一種のポップアップ・メニュー

プルダウン，ポップアップ，ワークエリア・メニューは同じ Menu widget クラスの widget として作成されます。また，メニュー・バーからプルダウン・メニューを出現させるための特別なプッシュ・ボタンとして Pull-Down Menu Entry widget を用います。サブ・メニューはプルダウン・メニューやポップアップ・メニューの項目としてプルダウン・メニューを指定することによって作成します。

Menu Widget は次の関数で作成します。

```
DwtMenu (parent_widget, name, x, y, format,
                orientation, entry_callback, map_callback, help_callback)
DwtMenuCreate (parent_widget, name,
                override_arglist, override_argcount)
DwtMenuPulldownCreate (parent_widget, name,
                        override_arglist, override_argcount)
DwtMenuPopupCreate (parent_widget, name,
                        override_arglist, override_argcount)
```

DwtMenu（高レベル関数）では format でメニューの形式を指定します。これには DwtMenuWorkArea, DwtMenuPulldown, DwtMenuPopup があります（この 3 つと Radio Box , Menu Bar はすべて Menu widget クラスに属しますが，インスタンスの初期化の方法が異なるため作成関数を区別しています）。DwtMenuCreate は Work Area Menu を，DwtDwtMenuPulldownCreate は Pull-Down Menu を，DwtMenuPopupCreate は Pop-Up Menu を作成します。サブ・メニューの作成は他の Menu のサブ widget として Pull-Down Menu を作成することで行ないます。Menu widget のおもなリソースを以下に示します。

DwtNorientation	メニューの項目を縦（DwtOrientationVertical）に並べるのか横（DwtNOrientationHorizontal）に並べるのかの指定。デフォルトは DwtOrientationVertical
DwtNentryCallback	サブ widget の DwtNactivateCallback に置き換えるコールバック
DwtNmenuRadio	ひとつの項目が選ばれる（on）とすでに選ばれている項目を自動的に off にするかどうかの指定（Radio Box で使用）
DwtNradioAlwaysOne	常にひとつだけの項目が選ばれているようにする（Radio Box で使用）
DwtNmenuHistory	前に選ばれた項目の widget ID を保持するために用いる (Pop-up Menu)，またはあらかじめ選ばれている項目を指定する（Radio Box）

DwtNentryCallback はサブ widget のコールバックの処理を 1 つのコールバック・プロシージャでまとめて行なうために使います。コールバック構造体は次の通り。

```
typedef struct {
   int  reason;
   XEvent  *event;
   Widget  s_widget;
   char   *s_tag;
   char   *s_callbackstruct;
} DwtMenuCallbackStruct;
```

Pop-up Menu widget の位置決めを MB2 の押された位置に基づいて行なうために次の関数が用意されています。

```
DwtMenuPosition (position, event)
```

position に Pop-up Menu widget の widget ID を，event にボタン・イベントを指定します。そして，この関数を呼んだ後，Pop-up Menu widget を XtMangeChild します。

Pull-Down Menu Entry widget はメニュー・バーのサブ widget，プルダウン・メニューのペアレント widget として作成し，プルダウン・メニューを出現させるために用います。スーパー・クラスは Label widget です。次の関数で作成します。

```
DwtPullDownMenuEntry (parenet_widget, name, x, y, label,
                      menu_id, callback, help_callback)
DwtPullDownMenuEntryCreate (parent_widget, name,
                            override_arglist, override_argcount)
```

menu_id (DwtNsubMenuId) にプルダウン・メニューを指定します。callback は DwtNactivateCallback (MB1 が押された) と DwtNpullingCallback (プルダウン・メニューの DwtNmapCallback の直前) の両方にセットされます。

Option Menu widget (スーパー・クラス Menu widget) は次の関数で作成します。

```
DwtOptionMenu (parent_widget, name, x, y, label,
               entry_callback, help_callback)
DwtOptionMenuCreate (parent_widget, name,
                     override_arglist, override_argcount)
```

作成された widget のサブ widget としてポップアップ・メニューを作成します。ボタンを押さない間はラベル (DwtNlabel) と，選択された項目が表示されます。

Separator widget はメニューの項目を区切って分類するための点線です。スーパー・クラスは Label widget です。widget 作成関数は

```
DwtSeparator (parent_widget, name, x, y, orientation)
DwtSeparatorCreate (parent_widget, name,
                    override_arglist, override_argcount)
DwtSeparatorGadgetCreate (parent_widget, name,
                          override_arglist, override_argcount)
```

です。orientation（DwtNorientation）で点線の縦横を指定します。

7.6. ダイアログ・ボックス widget

ポップアップ・ダイアログ・ボックス（または単にダイアログ・ボックス）はアプリケーションのメイン・ウィンドウとは独立したトップレベル・ウィンドウを作り，ユーザに対してメッセージを出力したり，それへの返答を得たりするための widget です。

Modal ダイアログ・ボックスはアプリケーションの処理を中断して，ユーザからの入力を受け付けます。Modeless ダイアログ・ボックスはアプリケーションの処理と並行して，ユーザとのやり取りを行ないます。Modeless ダイアログ・ボックスにはタイトル・バーが付きますが，Modal ダイアログ・ボックスには付きません。ダイログ・ボックスの表示を消すには "OK" や "Cancel" などのプッシュ・ボタンを押します。

ダイアログ・ボックスはラベルやプッシュ・ボタン，テキスト widget などのサブ widget を並べて表示します。サブ widget の位置の指定は通常のピクセル単位の指定の他，デフォルトのフォントを元にした単位による指定，サブ widget 間の位置関係による指定（Attached Dialog Box）が行なえます。また，ダイアログ・ボックスはデフォルト・プッシュ・ボタン（二重枠のプッシュ・ボタンで，Return キーを入力するとそのボタンが起動する）の設定や，テキスト入力エリア間の Tab キーによる移動の設定を行ないます。

Dialog Box widget の widget の作成関数は次の通り。

```
DwtDialogBox (parent_widget, name,
              default_position, x, y, title, style,
              map_callback, help_callback)
DwtDialogBoxCreate (parent_widget, name,
                    override_arglist, override_argcount)
DwtDialogBoxPopupCreate (parent_widget, name,
                         override_arglist, override_argcount)
```

style（リソース DwtNstyle）に DwtModal（Modal Dialog Box），DwtModeless（Modeless Dialog Box）を指定します（DwtDialogBoxPopupCerate）。この指定にはこれ以外にもうひとつ DwtWorkarea があり，ワーク・エリアに常に表示させておく Dialog Box を作ります（DwtDialogBox, DwtDIalogBoxCreate）。おもなリソースを以下に示します。

DwtNunits	サブ widget の x, y の位置指定の単位。DwtFontUnits（デフォルト）でフォントの大きさを元にした座標，DwtPixelUnits でピクセル単位となる

DwtNdefalutPosition	True にすると Dialog Box の位置がペアレント widget の真ん中に来るように設定される。デフォルトは False で，DwtNx，DwtNy に従う。
DwtNchildOverlap	サブ widget が重なることを許すかどうか。デフォルト True
DwtNresize	サブ widget が加わったりリサイズした時に Dialog Box の大きさをどうするかの指定。DwtResizeFixed で Dialog Box の大きさは変化しない，DwtResizeGrowOnly（デフォルト）で大きくなる場合だけリサイズする，DwtResizeShrinkWrap はサブ widget の大きさに合わせて大きさを変える

DwtFontUnits の x 方向の単位はフォント（DwtNfont の指定）の幅（QUAD_WIDTH プロパティの値，この値はそのフォントの数字の文字("0"～"9")の幅の 2 倍(おおざっぱには "M" の幅）で 10point のフォントで約 5point=5/72inch）の 1/4，y 方向は同じくフォントの幅の 1/8 がとられます。

DwtNbackground，DwtNforeground，DwtNborder，DwtNfont などのリソースはサブ widget として作成される gadget と共有します（gadget にはこれらのリソースがないため）。

また，Pop-Up Dialog Box widget では次のリソースも利用できます。

DwtNtitle	タイトル・バーに表示するタイトル
DwtNnoResize	そのウィンドウをリサイズできるかどうか(リサイズ・ボタンを付けるかどうか）の指定。デフォルトは True（リサイズできない）
DwtNautoUnmanage	デフォルト・プッシュ・ボタンやキャンセル・プッシュ・ボタンを押した時にPop-Up Dialog Box を自動的に消すかどうかの指定。デフォルトは True
DwtNdefaultButton	Return キーを押した時に起動するデフォルトのプッシュ・ボタン。そのプッシュ・ボタンは二重枠となる
DwtNcancelButton	キャンセル・プッシュ・ボタンの指定。Shift Return で起動する

Attached Dialog Box widget は通常の Dialog Box の機能に加えて，サブ widget の位置指定を widget 間で相対的に行なえるようにしています。widget 作成関数は次の通り。

```
DwtAttachedDB (parent_widget, name,
               default_position, x, y, title, style,
               map_callback, help_callback)
DwtAttachedDBCreate (parent_widget, name,
                     override_arglist, override_argcount)
DwtAttachedDBPopupCreate (parent_widget, name,
                          override_arglist, override_argcount)
```

style（DwtNstyle）の設定は Dialog Box widget と同様です。

Attached Dialog Box widget では Dialog Box widget のものに加えて次のリソースを利用できます。

DwtNdefaultHorizontalOffset, DwtNdefaultVerticalOffset
: サブ widget 間のデフォルトの間隔。単位は DwtNunits に従う

DwtNrubberPositioning
: サブ widget の attachment が省略された場合の処理の指定。False（デフォルト）で左と上を元の位置に attach する。True で上下左右を元の位置に attach する（Attached Dialog Box がリサイズするとサブ widget もそれにつれてリサイズする）

DwtNfractionBase
: DwtAttachPosition の分母となる数。デフォルト 100。DwtAttachPosition で指定する位置はこの数に対する比を用いる。デフォルトでは真ん中は 50 となる

位置の指定のためにサブ widget の方に指定するリソース（constraint リソース，Attached Dialog Box は constraint widget として機能する）は次の通り。

DwtNadbTopAttachment, DwtNadbBottomAttachment, DwtNadbLeftAttachment, DwtNadbRightAttachment
: widget のそれぞれの端の位置の決め方の指定。次の指定がある

　DwtAttachNone
　: 特に指定しない。他の widget の attachment か Attached Dialog Box の DwtNrubberPositioning の設定に従う

　DwtAttachAdb
　: Attached Dialog Box の（近い側の）ボーダーに attach する

　DwtAttachOppAdb
　: Attached Dialog Box の反対側（遠い側）のボーダーに attach する

　DwtAttachWidget
　: 他の widget（近い側のボーダー）に attach する。widget は DwtNaTopWidget などに指定する

　DwtAttachOppWidget
　: 他の widget の反対側（遠い側）のボーダーに

attach する

DwtAttachPosition	DwtNTopPosition などで指定した位置に attach する
DwtAttachSelf	自分自身の最初あった位置に attach する

DwtNadbTopWidget, DwtNadbBottomWidget, DwtNadbLeftWidget, DwtNadbRightWidget
DwtAttachWidget と DwtAttachOppWidget で attach する widget の指定

DwtNadbTopPosition, DwtNadbBottomPosition, DwtNadbLeftPosition, DwtNadbRightPosition
DwtAttachPosition で attach する位置の指定。この指定は DwtNfractionBase に対する比で（相対的に）行なう。Attached Dialog Box がリサイズするとそれに比例して（絶対的な）位置も移動する

DwtNadbTopOffset, DwtNadbBottomOffset, DwtNadbLeftOffset, DwtNadbRightOffset
実際に attach したい点の Attached Dialog Box（DwtAttachAdb）や widget（DwtAttachWidget）や位置（DwtAttachPosition）からの隔たり。デフォルトは DwtNdefalutHorizontalOffset または DwtNverticalOffset。単位は DwtNunit に従う

7.7. テキスト widget

Simple Text widget はテキスト・ストリングをキーボードとマウスを使って編集するための widget です。これは単に長いテキストを表示したり，ファイル名などユーザからの短い入力を受付けたり，テキスト・エディタとして利用したりすることができます。Simple Text widget は標準ダイアログ・ボックスの多くの場所で用いられています。

widget 作成関数は

```
DwtSText (parent_widget, name, x, y, cols, rows, value)
DwtSTextCreate (parent_widget, name,
                override_arglist, override_argcount)
```

です。おもなリソースを次に示します。

DwtNcols	表示するテキストのカラム数。デフォルト 20
DwtNrows	表示するテキスト行数。デフォルト 1

DwtNvalue　表示する，またはエディットしたテキストの内容。型は char *

DwtNtopPosition　表示を開始する位置（先頭からの文字数）

DwtNwordWrap　テキストの行の終わりの単語で自動的に折り返して表示するかどうかの指定。デフォルトは False で行の終わりは隠れてしまう

DwtNscrollVertical　縦方向のスクロール・バーを付けるかどうか。デフォルト False

DwtNresizeHeight, DwtNresizeWidth
テキストの大きさが変化したときに自動的に widget の大きさを合せるかどうか。デフォルト True

DwtNeditable　表示したテキストをエディットできるかどうか。デフォルト True

DwtNmaxLength　テキストの最大の長さ。デフォルト 2^31-1

DwtNinsertionPointVisible
文字の入力位置（カーソル）を表示するかどうか。デフォルト True

DwtNautoShowInsertPoint
カーソルが移動したときに自動的にスクロールするかどうか。デフォルト True

DwtNinsertionPosition
テキストの入力位置の指定

DwtNhalfBorder　ボーダーの表示を左下だけにする。デフォルト False。ダイアログ・ボックスのテキスト入力エリアで良く用いる

Simple Text widget を操作するために次のような関数が用意されています。

o テキスト（DwtNvalue）の参照と設定
```
DwtNSTextGetString (widget)
DwtNSTextSetString (widget, value)
```

o セレクトされたテキストの操作
```
DwtNSTextClearSelection (widget, time)
DwtNSTextGetSelection (widget)
DwtNSTextSetSelectin (widget, first_pos, last_pos, time)
```

ここで time はセレクションを取得する時間の指定です。キー・イベントやボタン・イベントで返されたものや CurrentTime を指定します

o テキストの置き換え
DwtNSTextReplace (widget, from_pos, to_pos, value)

o その他の関数
DwtNSTextGetMaxLength (widget)
DwtNSTextSetMaxLength (widget, max_length)
DwtNSTextGetEditable (widget)
DwtNSTextSetEditable (widget, editable)

Simple Text Widget のエディット機能のキー定義やマウスの定義をトランスレーション・テーブルの形で行なうことができます。利用できるアクションについては ULTRIX Reference Pages またはオンライン・マニュアルの X(1) を参照してください。

List Box widget は複数（一般に多数）の項目を表示し，そのうちのひとつまたは複数を選択するための widget です。スーパー・クラスのスクロール・ウィンドウの機能を用いて表示を行なっています。widget 作成関数は

```
DwtListBox (parent_widget, name, x, y,
                items, items_count, visible_items_count,
                collback, help_callback, resize, horizontal)
DwtListBoxCreate (parent_widget, name,
                        override_arglist, override_argcount)
```

です。callback に渡したコールバックは DwtNsingleCallback, DwtNsingleconfirmCallback, DwtNextendCallback, DwtNextendConfirmCallback に設定されます。おもなリソースは次のとおり。

DwtNitems, DwtNitemsCount
: 選択すべき項目とその数。同じ項目を複数設定することはできません。DwtNitems は DwtCompString の配列で指定する

DwtNvisibleItemsCount
: 表示する項目の数

DwtNsingleSelection
: 複数の項目を選べるかどうかの指定。デフォルトは True で 1 つしか選べない

DwtNresize
: 項目の長さに合わせて List Box の幅を変化させるかどうか。デフォルト True。False にする場合は DwtNhorizontal を True にすると良い

DwtNhorizontal
: 横向きのスクロール・バーを表示するかどうか。デフォルト False

DwtNselecetedItems, DwtNselectedItemsCount
: 選択された項目とその数。複数の項目を選択するには Shift MB1 で行なう

DwtNsingleCallback　1 つの項目が選ばれた（MB1 のクリック）ときに呼ばれるコールバック

DwtNsingleConfirmCallback
1 つの項目がダブル・クリックで選ばれたときに呼ばれるコールバック。アプリケーションは通常，ダブル・クリックで "OK" のボタンを押したときの処理を行なう

DwtNextendCallback　2 つめ以降の項目が選ばれたときに呼ばれるコールバック

DwtNextendConfirmCallback
2 つめ以降の項目がダブル・クリックで選ばれたときに呼ばれるコールバック。

List Box widget を操作するために次の関数が用意されています。

o 項目の追加，削除

```
DwtListBoxAddItem (widget, item, position)
DwtListBoxDeleteItem (widget, item)
DwtListBoxDeletePos (widget, position)
DwtListBoxItemExists (widget, item)
```

項目の番号（position）は 1 から数えます。DwtListBoxAddItem で position に 0 を指定すると項目の最後に付け加えられます。たくさんの項目を設定する場合はを DwtNitems と DwtNitemsCount を直接設定することもできます。ただし，この場合には DwtNselectedItems と DwtNselectedItemsCount の値と矛盾しないように気を付ける必要があります。DwtListBoxDeleteItem は項目の実体（DwtCompString）による指定，DwtListBoxDeletePos の番号による指定です

o セレクションの設定

```
DwtListBoxSelectItem (widget, item, notify)
DwtListBoxDeselectItem (widget, item)
DwtListBoxSelectPos (widget, position, notify)
DwtListBoxDeselectPos (widget, position)
DwtListBoxDeselectAllItems (widget)
```

選択された項目は DwtNselectedItems と DwtNselectedItemsCount に入ります。notifry はコールバックを呼ぶかどうかの指定です

o 項目の表示位置の設定

```
DwtListBoxSetItem (widget, item)
DwtListBoxSetPos (widget, position)
DwtListBoxSetHorizPos (widget, position)
```

7.8. 標準ダイアログ・ボックス widget

多くのアプリケーションはファイルをオープンするとか，メッセージを出すなどの共通の処理を行ないます。XUI Style Guide の定める標準的なダイアログ・ボックスの内，XUI Toolkit が widget として提供しているものを以下に示します。

Help widget は XUI の主要な特徴ともなっているオンライン・ヘルプを引くための widget です。Help widget はタイトル・バー，メニュー・バー，トピックの内容を表示するエリア，関連する他のトピックを呼び出すためのメニュー・エリア，前のトピックへ戻るためのプッシュ・ボタン，終了のプッシュ・ボタンから成るダイアログ・ボックスです。ヘルプ・トピックの検索はトピックの階層構造やキーワードやタイトルで行なうことができます。ヘルプ・トピックの内容はヘルプ・ファイルの形で行ないます。

widget 作成関数は次の通り。

```
DwtHelp (parent_widget, name,
                defalut_position, x, y, application_name,
                library_type, library_spec, first_topic,
                overview_topic, glossary_topic, unmap_callback)
DwtHelpCreate (parent_widget, name,
                override_arglist, override_argcount)
```

おもなリソースは次の通り。

DwtNapplicationName	タイトル・バーに表示するアプリケーションの名前。DwtCompString で指定する
DwtNlibraryType	ライブラリ・ファイルの形式の指定。DwtTextLibrary と指定する
DwtNlibrarySpec	ライブラリ・ファイルのあるディレクトリの名前。DwtCompString で指定する
DwtNfirstTopic	manage した時に最初に表示するトピックのファイル名。DwtCompString で指定する（以下同様）
DwtNoverviewTopic	アプリケーションの概要を説明したトピックのファイル名
DwtNglossaryTopic	用語の説明をしたトピックのファイル名

ヘルプ・ファイルは DwtNlibrarySpec で指定したディレクトリに置いておきます。フル・パス名は DwtNlibrarySpec に DwtNoverviewTopic や DwtNglossaryTopic の指定を付加したものになります。

トピックは階層構造を持ってヘルプ・ファイルに入ります。これは例えば次のように指定します（Help widget 自身のヘルプから引用）。

```
1 Overview
=Title Help on Help
=KEYWORD overview
This is the Help-on-Help overview frame. If you need
general advice on using Help, select "A beginner's
guide" from the list below. For advice on a specific ...

2 Over_guide_functions
=Title A beginner's guide
=KEYWORD help
The Help system performs in much the same way as other
DECwindows applications, so learning how Help behaves ...

3 accelerators
=Title The accelerator key
=KEYWORD accelerators
In common with other DECwindows applications, double
clicking mouse button 1 (MB1) when you are using Help ...
```

1 桁目の数字でトピックの階層レベルを指定します。そのトピックの 1 つ下のトピックが関連するトピックとして下のメニューに表示されます。"=TITLE string" でトピックのタイトル名を指定します。"=keyword string" でそのトピックに関連するキー・ワードを指定します。=TITLE と =KEYWORD の指定は省略することができます。また，"=INCLUDE topic title" で他の場所で記述されているトピックをそのトピックの関連するトピックに付け加えることができます。

Help widget の起動はヘルプ・メニュー，ヘルプ・キー，ヘルプ・コマンドまたはヘルプ・キーを押しながら MB1 をクリックして行なうように設定します。ヘルプ・メニュー以外の起動ではその場に応じたトピックを表示するようにします（context sensitive help）。

Work-in-Progress Box widget は時間のかかる処理の実行中であることをユーザに示すための widget です。この widget はメッセージを表示するためのラベルと処理を中止するためのプッシュ・ボタンからなるダイアログ・ボックスです。widget 作成関数は

```
DwtWorkBox (parent_widget, name, default_position, x, y,
                style, label, cancel_label,
                callback, help_callback)
DwtWorkBoxCreate (parent_widget, name,
                        override_arglist, override_argcount)
```

です。style（DwtNstyle）は Modal, Modeless の指定，label（DwtNlabel）はメッセージ，cancel_label（DwtNcancelLabel）はキャンセル・プッシュ・ボタンのラベル（デフォルト "Cancel"），callback（DwtNcancelCallback）はキャンセル・プッシュ・ボタンが押されたときに呼ばれるコールバック（reason は DwtCRCancel）です。

Message Box widget はユーザにメッセージを表示するためのダイアログ・ボックスです。この widget はメッセージのためのラベルとユーザがメッセージを確認したことを知らせるためのプッシュ・ボタンを持ちます。widget 作成関数は

```
DwtMessageBox (parent_widget, name, default_position, x, y,
                style, ok_label, label, callback, help_callback)
DwtMessageBoxCreate (parent_widget, name,
                        override_arglist, override_argcount)
```

です。ok_label (DwtNokLabel) のデフォルトは "Acknowledged" です。プッシュ・ボタンが押されると callback (DwtNyesCallback) が呼ばれ, Message Box widget は自動的に XtUnmanageChild されます。

Caution Box widget はエラー・メッセージを表示しユーザに対応を問い合わせるための widget です。"Yes", "No", "Cancel" の 3 つのプッシュ・ボタンを持ちます。widget 作成関数は次の通り。

```
DwtCautionBox (parent_widget, name, defalut_position, x, y,
                style, label,
                yes_label, no_label, cancel_label,
                default_push_button, callback, help_callback)
DwtCautionBoxCreate (parent_widget, name,
                        override_arglist, override_argcount)
```

default_push_button (DwtNdefaultPushbutton) はデフォルト・プッシュ・ボタン (Return で起動する) の指定で, DwtYesButton, DwtNoButton, DwtCancelButton のいずれかを指定します。callback は DwtNyesCallback, DwtNnoCallback, DwtNcancelCallback の 3 つのコールバックの設定です (reason はそれぞれ DwtCRYes, DwtCRNo, DwtCRCancel)。

Command Window widget はコマンド入力エリアを作成するための widget です。Command Window widget はコマンドを入力するための入力行と, ヒストリを表示するためのエリアを持ちます。ヒストリは上下のカーソル・キーで呼び出すことができます。widget 作成関数は次の通り。

```
DwtCommandWindow (parent_widget, name,
                        prompt, lines, callback, help_callback)
DwtCommandWindowCreate (parent_widget, name,
                        override_arglist, override_argcount)
```

prompt (DwtNprompt) はコマンド行に表示するプロンプト (DwtCompString), lines (DwtNlines) はヒストリ・エリアの行数, callback は Retrun が入力されたときに呼ばれるコールバック (DwtNcommandEnteredCallback) とコマンド行の内容が変更されたときに呼ばれるコールバック (DwtNvalueChangedCallback) の設定です。コールバック構造体は次の通り。

```
typedef struct {
   int  reason;
   XEvent  *event;
   int  length;
   char  *value;
} DwtCommandWindowCallbackStruct;
```

ヒストリ・エリアの内容は DwtNhistory に保持されています。Command Window widget を操作するために次の関数が用意されています。

```
DwtCommandAppend (widget, command)
DwtCommandSet (widget, command)
DwtCommandErrorMessage (widget, message)
```

DwtCommandAppend はコマンド行に指定した文字列を連結します。DwtCommandSet はコマンド行の内容を置き換えます。どちらも文字列中に <CR> または <LF> または <CR><LF> があると DwtNcommandEnterdCallback が呼ばれます。DwtCommandErrorMessage はヒストリ・エリアにメッセージを表示します。

Selection Box widget はリスト・ボックスを用いて 1 つの項目を選択するための標準的なダイアログ・ボックスです。"OK" と "Cnacel" のプッシュ・ボタンと選ばれた項目を表示するテキスト・エリアを持ちます。widget 作成関数は

```
DwtSelection (parent_widget, name, x, y, title,
                value,
                items, items_count, visible_items_caount,
                style, defalt_position, callback, help_callback)
DetSelectinCreate (parent_widget, name,
                        override_arglist, override_argcount)
```

です。value (DwtNvalue) はテキスト・エリアに表示する項目，すなわち選択された項目です (DwtCompString) 。callback は DwtNactivateCallback, DwtNcancelCallback, DwtNnoMatchCallback の 3 つのコールバックを設定します。DwtNnoMatchCallback は items (DwtNitems) の項目にないテキストを入力した場合に呼ばれます (DwtNmustMatch が False の場合(デフォルト)) 。

File Selection widget はファイルを検索し，その名前を入力するための widget です。ワイルド・カードで検索条件を指定するエリア，条件に合ったファイル名を表示するリスト・ボックス，選択したファイル名を表示するテキスト・エリアから成っています。

スーパー・クラスは Selection Box widget です。widget 作成関数は次のとおり。

```
DwtFileSelection (parent_widget, name, x, y, title,
                        value, dirmask, visible_items_count,
                        style, default_position,
                        callback, help_callback)
DwtFileSelectionCreate (parent_widget, name,
                        override_arglist, override_argcount)
```

value (DwtNvalue) は選択されたファイル名，dirmask (DwtNdirMask) は検索する条件の指定です (どちらも DwtCompString) 。callback は DwtNactivateCallback, DwtNcancelCallback, DwtnnoMatchCallback の設定です。コールバック構造体は

```
typedef struct {
   int  reason;
   XEvent  *event;
   DwtCompString  value;
   int  value_len;
   DwtCompString  dirmask;
   int  dirmask_len;
} DwtFileSelectionCallback;
```

です。

7.9. コンパウンド・ストリング関数

コンパウンド・ストリングは DEC の定める CDA (Compound Document Architechture) に基づく文字列の表現形式です。XUI Toolkit の Label widget を始め多くの widget はこの形式を用いて文字列を表現しています。コンパウンド・ストリングは 1 つの文字列に複数の種類のフォントや属性を持つ文字を含ませることができます。属性には国籍情報（どこの国の文字か），テキストの標記方向，修飾情報，キャラクタ・セット情報（文字コードと字形の対応）があります。

7.9.1. フォント・リスト

XUI Toolkit は複数のフォントを使用するため，キャラクタ・セットとフォントをフォント・リストで対応付けます。フォント・リストは次の関数で作成します。

```
DwtFontList DwtCreateFontList (font, charset)
   XFontStruct  *font;
   long  charset;

DwtFontList DwtAddFontList (list, font, charset)
   DwtFontList  list;
   XFontStruct  *font;
   long   charset;
```

XFontStruct は Xlib の関数の XQueryFont で得ます。キャラクタ・セット（charset）は <cda_def.h> で定義されています。英語の通常の文字は CDA$K_ISO_LATIN1，漢字は CDA$K_DEC_KANJI です。

フォント・リストは各 widget の DwtNfont というリソースに設定します。UIL でフォント・リストを作成するには FONT_TABLE 関数を用います。

7.9.2. コンパウンド・ストリングの作成

コンパウンド・ストリングは次の関数で作成します。

```
DwtCompString DwtString (text, charset, direction_r_to_l)
   char  *text;
   unsigned long  charset;
   Boolean  direction_r_to_l;
```

direction_r_to_l を True にするとテキストは右から左に表示されます。また，国籍情報と修飾情報は現在使われていないので設定する必要はありません（設定するためには DwtCSString (text, charset, direction_r_to_l, language, rend) を使う）。

LATIN1 の文字セットの文字列を作成する場合は次の関数を用いることができます。

```
DwtCompString DwtLatin1String (text)
   char  *text;
```

複数の文字セットを含んだ文字列を作成する場合にはそれぞれの部分を次の関数で連結して作成します。

```
DwtCompString DwtCStrcat (compound_string1, compound_string2)
   DwtCompString  compound_string1, compound_string2;

DwtCompString DwtCStrncat (compound_string1, compound_string2,
                           num_chars)
   DwtCompString  compound_string1, compound_string2;
   int  num_chars;
```

どちらの関数も compound_string2 を compound_string1 の後ろに連結します。

これらの関数で作成したコンパウンド・ストリングの領域は XtFree で解放します。

コンパウンド・ストリングは Label widget のラベル（DwtNlabel）などで使用します。UIL では "text" や #charset"text", "text1" & "text2" などでコンパウンド・ストリングを作成することができます。

7.9.3. コンパウンド・ストリングの操作

コンパウンド・ストリングの比較は次の関数で行ないます。

```
int DwtCSbytecmp (compound_string1, compound_string2)
   DwtCompString  compound_string1, compound_string2;
```

DwtCSbytecmp は文字列を 1 バイト毎に比較して等しければ 0 を，そうでなければ 1 を返します。コンパウンド・ストリングの長さを知るには

```
int DwtCStrlen (compound_string)
   DwtCompString  compound_string;
```

を用います。

コンパウンド・ストリングに含まれる文字を C 言語の文字列として得るには次の関数を用います。

```
int DwtInitGetSegment (context, compound_string)
   DwtCompStringContext  *context;
   DwtCompString  compound_string;

int DwtGetNextSegment (context, text_retrun,
                       charset_return, direction_r_to_l_return,
                       lang_return, rend_return)
   DwtCompStringContext  context;
   char  *text_return;
   long  *charset_return;
   Boolean  *direction_r_to_l_return;
   long  *lang_return;
   long  *rend_return;
```

DwtInitGetSegment は渡した context を初期化します。DwtGetNextSegment は次のセグメント（同じ属性を持つ部分文字列）を返します。それぞれの戻り値は DwtSuccess, DwtFail（compound_string または context が有効でない），DwtEndCS（compound_string が NUL または，もうセグメントがない）です。

7.10. XUI Toolkit widget を用いたプログラミング

XUI Toolkit は数多くの高機能な widget を提供しています。また，高レベルな widget 作成関数を用いると ArgList の設定などの繁雑な処理が最小限に押さえられます。さらに，後に述べる UIL を用いることによって個々の widget を作成するための処理さえ不要となります。

XUI Toolkit の widget を用いるプログラムではヘッダファイルのインクルードのために

```
#include <X11/DwtAppl.h>
```

の宣言を行ないます。コンパイル時のコマンドとしては

```
cc file.c -ldwt -lX11
```

のように指定します。

XUI Toolkit の Push Button widget を用いた "Hello World!" プログラムを示します。

```
/*
 * hw_xui.c
 *      sample implementation of hw on XUI
 *      to compile this, enter;
 *              % cc cc hw_xui.c -ldwt -lX11
 *
 * Tat001       18-Jul-1989     for x11pg
 */

#include <X11/DwtAppl.h>

static char     *hello_world = {"Hello world!"};
void hw_callback();

main (argc, argv)
int     argc;
char    *argv[];
{
        Widget  toplevel, hw;
        static DwtCallback hw_callbacks[] = {{hw_callback, 0}, {0, 0}};

        /* initialize toolkit and create top-level shell */
        toplevel = XtInitialize ("hw on XUI", "Test", 0, 0, &argc, argv);

        /* create push button */
        hw = DwtPushButton (toplevel, "hw",
                        0, 0, DwtLatin1String(hello_world),
                        hw_callbacks, 0);
        XtManageChild (hw);

        XtRealizeWidget (toplevel);

        XtMainLoop ();
}

/* push button callback procedure */
void hw_callback (widget, tag, reason)
Widget  widget;
int     tag;
DwtAnyCallbackStruct    *reason;
{
        exit ();
}

/* end of hw_xui.c */
```

7.11. DRM

DRM (XUI Resource Manager) は XUI Toolkit の提供する，XUI のためのリソース・マネージャです。DRM は UIL コンパイラの作る UID ファイルを元に widget の作成を行ないます。また，UID が参照しているシンボルをプログラムから登録したり，UID で定義した値の参照などを行なうこともできます。

7.11.1. DRM の初期化

DRM の初期化には次の関数を用います。

```
void DwtInitializeDRM ()
```

DwtInitializeDRM は XtInitialize と DwtRegisterClass よりも先に呼び出す必要があります。

7.11.2. UID のロードとデータ階層の初期化

UID ファイルをロードし，DRM のデータ階層 (search hierarchy) を初期化するには

```
Cardinal DwtOpenHierarchy (num_files, file_names_list,
                           ancillary_structures_list,
                           hierarchy_id_return,)
    DRMCount  num_files;
    String  *file_names_list;
    IDBOSOpenParamPtr  *ancillary_structures_list;
    DRMHierarchy  *hierarchy_id_return;
```

num_files, file_name_list	UID ファイルのファイル指定 (パス名) の配列
ancillary_structures_list	OS に依存した属性の指定。省略可能
hierarchy_id_return	DRM のデータ階層の ID が返る

を用います。戻り値は DRMSucess, DRMNotFound, DRMFailure のいずれかです。hierarchy_id_return は以降の DRM 関数でどのデータ階層を用いるのかを示すために使います。また，UID ファイルを複数指定することによってそれらに階層構造を持たせることができます。この場合，データは file_names_list で前に指定したファイルの内容から順に検索されます。

DRM のデータ階層を閉じるには

```
Cardinal DwtCloseHierarchy (hierarchy_id)
   DRMHierarchy  hierarchy_id;
```

を用います。

7.11.3. プログラムから DRM へのデータの登録

DRM を用いて実際に widget を作成するのに先立って，UID が参照しているシンボルやコールバック・プロシージャを DRM に登録する必要があります。また，アプリケーション内で定義した widget クラスの登録も同様に必要です。

UID のオブジェクト（widget）が参照しているコールバック・プロシージャ（UIL の PROCEDURE）とプログラム中のシンボル（IDENTIFIER）の登録は次の関数で行ないます。

```
Cardinal DwtRegisterDRMNames (register_list, register_count)
   DRMRegisterArglist  register_list;
   DRMCount  register_count;
```

DRMRegisterArglist は次のような構造体の配列です。

```
typedef struct {
   String  name;       /* case-sensitive name */
   caddr_t  value;     /* value/address associated with name */
} DRMRegisterArg, *DRMRegisterArglist ;
```

name に UID で参照しているシンボルの名前，value にその名前に対応付ける値を指定します。コールバック・プロシージャの場合は value はそのプロシージャのエントリ・ポイントのアドレスとなります。DwtRegisterDRMNames によって DRM は UID によるプログラム中のシンボルの参照を実行時に解決します。

アプリケーションで定義した，UIL コンパイラがサポートしていない widget クラスの登録は，次の関数で行ないます。

```
Cardinal DwtRegisterClass (class_code, class_name, create_name,
                                    create_proc, class_record)
   DRMType  class_code;
   String  class_name;
   String  create_name;
   Widget  (* create_proc) ();
   WidgetClass  class_record;
```

class_code	widget クラスのクラス・コード。アプリケーションが定義した widget では DRMwcUnknown を指定する
class_name	widget クラス名
create_name	widget インスタンスを作成する関数の名前。UIL の USER_DEFINED PROCEDURE で宣言した PROCEDURE 名に対応する。
create_proc	widget インスタンスを作成する関数
class_record	widget クラスのクラス・レコード

インスタンスを作成するための関数は次のような形式で用意する必要があります。これは，XUI Toolkit の低位の widget 作成関数と同じ形式です。

```
Widget create_proc (parent_widget, name,
                        override_arglist, override_argcount)
   Widget  parent_widget;
   char  *name;
   ArgList  override_arglist;
   int  override_argcount;
```

7.11.4. widget の作成 (fetch)

アプリケーションの Main Window と Main Window の widget ツリー（コントロール・リストの階層）に含まれる widget は DRM を用いて一度に作成することができます。これには

```
Cardinal DwtFetchInterfaceModule (hierarchy_id, module_name,
                                  parent_widget, widget_return,)
   DRMHierarchy  hierarchy_id;
   char  *module_name;
   Widget  parent_widget;
   Widget  *widget_return;
```

hierarchy_id	DRM のデータ階層の ID
module_name	トップ・レベルのオブジェクト (main_window) の定義を含むUIL のモジュール名
parent_widget	作成される widget のペアレント widget となる widget。通常は XtInitialize によって作られたシェル widgetl を指定する。
widget_return	DRM によって作られた widget が返される

を用います。Main Window Widget がない場合は widget の作成は行ないません。(戻り値は DRMNotFound)

UID から個々の widget を作成するには次の関数を用います。

```
Cardinal DwtFetchWidget (hierarchy_id, index, parent_widget,
                widget_return, class_return)
   DRMHierarchy  hierarchy_id;
   String  index;
   Widget  parent_widget;
   Widget  *widget_return;
   DRMType  *class_return;
```

hierarchy_id	DRM のデータ階層の ID
index	作成する widget の UIL のオブジェクト名
parent_widget	作成される widget のペアレントとなる widget
widget_return	作成された widget の ID が返る。呼び出し時には NULL を入れておく。
class_return	作成された widget の widget クラス・コードが返る。DRMwcMainWindow, DRMwcUnknown など

ひとつのオブジェクトに対して DwtFetchWidget を複数回呼び出して，同じ属性を持つインスタンスを複数個作成することもできます。また，DwtFetchWidget で作成しようとする widget は他の widget の widget ツリーに含まれていてはいけません。

DwtFetchInterfaceModule と DwtFetchWidget で作成した widget を起動（realize）するには，まず XtManageChild でペアレント widget の登録リストに加える必要があります。

また，UID で指定されたリソースの一部を，プログラム中で置き換えて widget を作成するために DwtFetchWidgetOverride (hierarchy_id, index, parent_widget, override_name, override_args, override_num_args, widget_return, class_return) という関数が用意されています。すでにある widget のリソースの値を UID で定義した値を用いて設定する DwtFetchSetValues (hierarchy_id, widget, args, num_args) という関数もあります。

7.11.5. UID で定義したシンボルの参照

UIL で定義したシンボル（VALUE）をプログラムから参照することができます。次の関数を用います。

```
Cardinal DwtDrmGetResourceContext (alloc_func, free_func,
                                   size, context_id_return)
   char  *((*alloc_func)());
   void  (*free_cunf)();
   DRMSize  size;
   DRMResourceContextPtr  *context_id_return;

DRMType DwtDrmRCSetType (context_id, type)
   DRMResourceContextPtr  context_id;
   DRMType   *type;

Cardinal DwtDrmHGetIndexedLiteral (hierarchy_id, index, context_id)
   DRMHierarchy  hierarchy_id;
   String  index;
   DRMResourceContextPtr  context_id;

char *DwtDrmRCButter (context_id)
   DRMResourceContextPtr  context_id;

Cardinal DwtDrmFreeResourceContext (context_id)
   DRMResourceContextPrt  context_id;
```

alloc_func, free_func
: resource context のためのバッファを確保，解放するための関数。NULL を指定すると XtMalloc と XtFree が使われる。

size
: resource context で用いるバッファの大きさ

context_id_return, context_id
: 確保された resource context の ID。これを用いて他の処理を行なう。

type
: DwtDrmGetIndexedLiteral で返されるデータの形式。RGMrTypeInteger, RGMrTypeChar8, RGMrTypeCString などがある。

index
: 参照するシンボルの名前。UIL の VALUE セクションで定義したもの

値の参照は resource context というデータ・ブロックを介して行ないます。これは DwtDrmGetResourceContext を呼ぶことによって確保されます。そして，DwtDrmHGetIndexedLiteral を呼び出すと index の値が DwtDrmRCSetType で設定した型に変換され、DwtDrmRCButter の領域に置かれます。

7.12. UIL

UIL (User Interface Language) はユーザ・インターフェースを記述するための一種のプログラム言語です。UIL を用いてユーザ・インターフェースに含まれるメニュー，ダイアログ・ボックス，ラベル，ボタンなどのオブジェクトの初期の状態と，ユーザの操作によりどのコールバック・プロシージャが起動されるのかを定義することができます。UIL を用いることによってユーザ・インターフェースの記述とアプリケーションの本来の処理を分離することができます。また，ユーザ・インターフェースのカスタマイズなども UIL を変更するだけで容易に行なえます。

UIL コンパイラは UIL ファイルを UID (User Interface Definition) ファイルにコンパイルします。プログラム中で DRM の関数を呼び出すことによってその UID ファイルを利用することができます。UID は DRM で利用するのに効率の良い形式になっています。プログラムの実行中，DRM はユーザ・インターフェースのその時点での状態を保持します。

UIL を用いて次に挙げることを定義することができます。

- o インターフェースの構築に用いるオブジェクト (widget および gadget)
- o そのオブジェクトの属性 (widget のリソース)
- o 各 widget のコールバック・プロシージャ
- o アプリケーション全体のオブジェクトの階層構造

これらの定義は UIL モジュールという形で UIL ファイルに記述します。UIL コンパイラは XUI Toolkit の widget に対して，属性やコールバック，サブ widget の正当性のチェックを行ないます。

7.12.1. モジュール

1 つの UIL ファイルは 1 つの UIL モジュール (インターフェース・モジュール) から成ります。UIL モジュールは複数のオブジェクト，オブジェクトの定義に用いる種々の属性，プログラムとのインターフェースに用いるシンボルなどを関連付けて定義します。また，プログラムから同時に複数の UIL モジュールを利用することもできます。UIL モジュールは次の形式で記述します。

```
MODULE module-name
        [version-clause]
        [case-sensitivity-clause]
        [default-character-set-clause]
        {include-directive |
         value-section |
         identifier-section |
         procedure-section |
         list-section |
         object-section}...
END MODULE;
```

version-clause はインターフェースのバージョン番号を文字列で記述します。case-sensitivity-clause はシンボル名の大文字，小文字を区別するかどうかの指定です。default-character-set-clause は文字列（Compound String）のデフォルトの文字セットを指定します。それぞれ，次のように指定します。

```
VERSION = character-expression
NAMES = {CASE_SENSITIVE | CASE_INSENSITIVE}
CHARACTER_SET = character-set
```

NAMES のデフォルトは CASE_INSENSITIVE，デフォルトの文字セット（CHARACTER_SET）のデフォルトは ISO_LATIN1 です。NAMES が CASE_INSENSITIVE の場合は定義したシンボルはすべて大文字に変換されて UID ファイルにストアされます。CASE_SENSITIVE の場合はモジュール名以外はそのままの形でストアされます。CASE_SENSITIVE の場合，UIL のキーワードは小文字で記述する必要があります。

7.12.2. 他のファイルのインクルード

include-directive は他の UIL ファイルの引用する際の指定で，複数のモジュールで共通の定義を利用するのに便利です。記述は

```
INCLUDE FILE character-expression;
```

とします。

7.12.3. データの宣言

value-section ではモジュールで用いるシンボルとその値を定義します。外部シンボルの参照や，大域シンボルの定義も行なうことができます。値の型には COLOR, FONT, STRING, REASON, TRANSLATION TABLE など widget の定義のために有用なものが用意されています（後述）。次のように記述します。

```
VALUE
   value-name : {EXPORTED value-expression |
                 PRIVATE value-expression |
                 value-expression |
                 IMPORTED value-type} ; ...
```

EXPORTED のシンボルは他のモジュールから参照できるシンボルです。PRIVATE のシンボルはそのシンボルを定義したモジュールでのみ有効なものです。IMPORTED は他のモジュールで，EXPORTED 定義されたシンボルを参照するためのものです。デフォルトは PRIVATE です。

また，VALUE によるシンボルの宣言，はそのシンボルが参照される前にしておく必要があります。

7.12.4. プログラム中の値の参照

identifier-section ではプログラムで定義した値を UIL から参照するための宣言を行ないます。形式は

```
IDENTIFIER
   identifier-name, ....
```

です。実際の値はプログラム中で DwtRegisterDRMNames を用いて登録します。

7.12.5. プロシージャの宣言

procedure-section では UIL から参照するプログラム中のプロシージャの宣言を行ないます。これには widget が呼びだすコールバック・プロシージャと，アプリケーションで定義した widget を作成するためのプロシージャがあります。

```
PROCEDURE
   procedure-name [([value-type, ...])]; ...
```

かっこ内はプロシージャの引数の定義です。これによって UIL コンパイラは引数のチェックを行ないます。省略するとチェックは行なわれません。プログラムによるプロシージャの登録は IDENTIFIER と同様，DwtRegisterDRMNames で行ないます。

7.12.6. 属性リストの宣言

list-section はオブジェクトの定義に用いる，属性リストをあらかじめ定義するための宣言です。この定義はオブジェクト定義の中で直接記述することもできます。フォーマットは

```
LIST
   list-name : list-definition; ...
```

です。

list-definition には ARGUMENTS リスト，CALLBACKS リスト，CONTROLS リストの 3 種類があります(後述)。

7.12.7. オブジェクトの定義

object-section でインターフェース・オブジェクト（widget および gadget）の定義を行ないます。フォーマットは

```
OBJECT
   object-name : {EXPORTED object-definition |
                  PRIVATE object-definition |
                  object-definition |
                  IMPORTED object-type} ;
```

です。IMPORTED は他の UIL モジュールの EXPORTED オブジェクトを利用する際に用います。デフォルトは PRIVATE です。

object-type はどの widget クラスのインスタンスを利用するかの指定です。これには次のものがあります。

```
attached_dialog_box, caution_box, command_window, dialog_box,
file_selection, help_box, label, list_box, main_window, menu_bar,
message_box, option_menu, popup_attached_db, popup_dialog_box,
popup_menu, pulldown_entry, pulldown_menu, push_button, radio_box,
scale, scroll_bar, scroll_window, selection, separator, simple_text,
toggle_button, window, work_area_menu, work_in_progress_box,
user_defined
```

これらは XUI Toolkit のそれぞれの widget に対応しています（ user_defined については後述）。

object-definition は次のように記述します。

```
object-type
   object-name [WIDGET | GADGET] |
   [WIDGET | GADGET] {list-definition ...}
```

object-name の指定で他のオブジェクトを参照することができます。object-type が

separator, label, push_button, toggle_button の場合は GADGET を指定して，それぞれの gadget インスタンスを利用することができます。また，この指定の代わりに

```
OBJECT = {object-type = [WIDGET | GADGET]; .....}
```

をモジュールの始めに書いてそれぞれのデフォルトを WIDGET から GADGET に変更することもできます。

list-definition には ARGUMENTS リスト，CALLBACKS リスト，CONTROLS リストの 3 種類があります。

7.12.7.1. ARGUMENTS リスト

ARGUMENTS リストは widget に渡す ArgList の指定です（コールバック・リストは除く）。形式は

```
ARGUMENTS
   list-name |
   {argument-name = value-expression; ...}
```

です。list-name は list-section で宣言したものを参照する場合の指定です。

argument-name には UIL コンパイラに組み込まれているものと，VALUE セクションで定義したもの（ARGUMENT 関数）を指定することができます。UIL コンパイラは指定された argument-name がそのオブジェクトにふさわしいものかどうか，値の型があっているかどうかをチェックします。また，items や selected_items の argument-name に対して，自動的に items_count や selected_items_count が生成されます。

7.12.7.2. CALLBACKS リスト

CALLBACKS リストは widget のコールバック・プロシージャの指定です。形式は

```
CALLBACKS
   list-name |
   {reason-name = PROCEDURE procedure-name [([value-expression])]; ...}
```

です。reason-name は UIL コンパイラ組み込みのものと VALUE セクションで REASON 関数を用いて定義したものを指定することができます。procedure-name はコールバック・プロシージャの名前で，あらかじめ PROCEDURE セクションで宣言しておく必要があります。

プログラム中でのコールバック・プロシージャは，次の様な形式で作成し，DwtRegisterDRMNames を用いて DRM に登録します。

```
void procedure-name (widget, tag, callback_data)
   Widget  widget;
   Opaque  tag;
   DwtAnyCallbackStruct  callback_data
```

引数の tag に UIL の CALLBACKS リストで引数として指定した値（上記 value-expression）が渡されます。callback_data はコールバック reason によって形式が違いますが，先頭には reason の ID が入っています。

7.12.7.3. CONTROLS リスト

CONTROLS リストではその widget のサブ widget を定義します。サブ widget はペアレント widget に制御（control）されます。形式は

```
CONTROLS
   list-name |
   {[MANAGED | UNMANAGED] object-definition; ...}
```

です。MANAGED の指定は DRM がサブ widget を XtManageChild するかどうかの指定です。デフォルトは MANAGED です。サブ widget として指定するオブジェクトは先に定義されていなくてもかまいません。

7.12.8. データ・タイプとその操作

UIL コンパイラは次に挙げるデータ・タイプをサポートしています。

ANY, ARGUMENT, BOOLEAN, COLOR, COLOR_TABLE, COMPOUND_STRING, FLOAT, FONT, FONT_TABLE, ICON, INTEGER, REASON, STRING, STRING_TABLE, TRANSLATION_TABLE

ANY はプロシージャの引き数に用いて型チェックを省略させるためのデータ・タイプ名です。以下に各型とその値に対する操作について述べます。

7.12.8.1. 数値

整数と実数の両方が扱えます。INTEGER と FLOAT とういう関数で双方の変換を行なうこともできます。整数の演算子には四則演算（"+", "-", "*", "/"）の他にシフト（">>", "<<"）とビット演算の "&", "|", "^"（XOR），"~"（NOT）が用意されています。

論理型の数値として TRUE, FALSE, ON, OFF のキーワードが用意されています。UID ファイル中では真（TRUE と ON）は 1，偽（FALSE と OFF）は 0 で表現されます。

7.12.8.2. 文字列

UIL では文字列（STRING 型）は "'" で囲んで記述します。C 言語と同様，特殊文字は "\n", "\\" のように "\" でエスケープします。また，"\nnn"（ nnn は 8 進の数値）としてキャラクタコードで指定することもできます。この文字列は null でターミネートした形で保持されます。

""" で囲んだ文字列は COMPOUND_STRING となります。文字セットは CHARACTER_SET で指定したものです。#char-set"char..." として別の文字セットを指定することもできます。関数，

```
COMPOUND_STRING (string-expressing
                 [,CHARACTER_SET=char-set] [,RIGHT_TO_LEFT=boolean]);
```

で作ることもできます。

文字列の連結は "&" で行ないます。

また，list_box の items と selected_items を指定するために STRING_TABLE が用意されています。STRING_TABLE は COMPOUND_STRING の配列です。次の関数で生成できます。

```
STRING_TABLE (character-expression ,...)
```

7.12.8.3. widget のリソース

ARGUMENT 型は ARGUMENTS リストの argument-name に用いるデータのデータ・タイプです。これは widget のリソース名に直接対応しています。XUI Toolkit の widget で使用するリソース名はすべて用意されています。これ以外のリソースを使用したい場合は ARGUMENT 型のデータを次の関数で生成して用います。

```
ARGUMENT (character-expression [,argument-type])
```

character-expressin がリソース名です。argument-type を省略すると ANY となります。この宣言は通常，VALUE セクションで

```
VALUE argument-name : ARGUMENT (resource-name, resource-type);
```

のように行ないます。

コールバック・リストのためのリソースには次の REASON を使用します。

7.12.8.4. コールバック・リスト

REASON は CALLBACKS リストの reason-name に用いるデータです。これは widget のコールバック・プロシージャのコールバック reason の名前に相当します。UIL コンパイラは reason-name で指定されたリソースに対してコールバック・リストを設定します。XUI Toolkit の widget のものは既に用意されています。新たにコールバック reason を追加する場合は

```
REASON (character-expression)
```

で生成します。character-expression はこの reason を使用するコールバック・リストのリソース名です。

UIL コンパイラは create という特別の REASON を用意しています。この REASON を用いて，コールバック・リストにコールバック・プロシージャ名を定義することにより，DRM によって自動的に作成された widget をプログラムから検出することができます（widget ID を記録したりする）。

7.12.8.5. 色

widget で使用する色（COLOR 型）は次の関数で生成します。

```
COLOR (string-expression [. FOREGROUND | BACKGROUND])
```

DRM は string-expression で指定した文字列を用いてカラー・マップの設定を行ないます。FOREGROUND と BACKGROUND はサーバがモノクロームの場合やカラーマップが一杯になった場合に，その色をどちらに対応付けるかの指定です。

ユーザの自由度と使い勝手の面から，色の指定は，インターフェース側で固定せずに .Xdefaults などのリソースファイルで行なうのが一般的です。

7.12.8.6. ピックスマップ

ICON 型はピックスマップを記述するためのデータ・タイプです。次の関数で ICON 型のデータを生成することができます。

```
ICON ([COLOR_TABLE=color-table-exp,] row ,...)
```

row は何種類かの文字を組み合わせた文字列で，ピックスマップの各行（scan line）に対応します。color-table-exp で row で用いる文字と色の対応を指定します。デフォルトの COLOR_TABLE はフォアグランドが "*"，バックグランドが " " です。

COLOR_TABLE は次の関数で生成します。

```
COLOR_TABLE (color-exp = letter ,...)
```

color-exp は COLOR 関数で宣言した色です。"FOREGROUND COLOR" と "BACKGROUND COLOR" のキーワードを指定することもできます。letter は ICON の指定に用いる文字で

す。

ICON の指定は例えば次のようになります。

```
value
   red         : color ('Red', foreground);
   arrow_color : color_table (red='o', background color=' ');
   arrow       : icon (color_table=arrow_color, '   o   ',
                                                '  ooo  ',
                                                ' ooooo ',
                                                'ooooooo',
                                                '   o   ',
                                                '   o   ',
                                                '   o   ');
```

7.12.8.7. フォント

フォントは次の関数で生成します。

```
FONT (string-expression [,CHARACTER_SET=cahr-set])
```

また，フォント・リストは次のように生成します。

```
FONT_TABLE ([char-set=]font-expression ,...)
```

これらによって，DRM は自動的にフォントのロードを行います。

7.12.8.8. トランスレーション・テーブル

widget で用いるトランスレーションテーブルは次のようにして生成することができます。

```
TRANSLATION_TABLE (character-expression ,...)
```

各 character-expression がトランスレーションの指定です。

7.12.9. アプリケーションで定義した widget クラスの利用

アプリケーションで定義した widget (UIL コンパイラがあらかじめ用意していないもの) を利用するには USER_DEFINED というオブジェクト・タイプを用います。

まず, widget インスタンスを作成する関数を PROCEDURE 宣言します。

```
PROCEDURE create-procedure;
```

このプロシージャは低位の widget 作成関数と同じ形式を持つもので, プログラム中で DwtRegisterClass で DRM に登録します。

オブジェクトの定義は次のようにします。

```
OBJECT object-name : USER_DEFINED PROCEDURE create-procedure
                         {list-definition ...};
```

また, このオブジェクトを ARGUMENTS リストや CONTROLS リストから参照するには

```
USER_DEFINED object-name
```

とします。

USER_DEFINED のオブジェクトではすべての argument-name とコールバック reason を指定することができます。UIL で用意されていないものについては VALUE セクションでそれぞれ宣言してやる必要があります。

7.13. UIL と DRM を用いたプログラミング

XUI Toolkit の UIL を用いると widget の作成がさらに簡単になります。UIL を用いるためにはプログラム上でいくつかの DRM ルーチンを呼び出します。また, UIL はプログラムと独立しているので, ユーザ・インターフェースの変更やカスタマイズがプログラムをいじらずに行なえます。

UIL と DRM を用いたプログラムでは XUI Toolkit の widget を直接用いるプログラムと同様,

```
#include <X11/DwtAppl.h>
```

の宣言を行ないます。プログラムのコンパイルも

```
cc file.c -ldwt -lX11
```

とします。

UIL のコンパイルは

```
dxuil file.uil -o file.uid
```

です。

UIL を用いた "Hello World!" プログラムを示します。DRM ルーチンによって widget が自動的に作成されます。UIL と DRM は複雑な親子関係を持った，たくさんの widget を用いるプログラムほど威力を発揮します。

```
/*
 * hw_uil.c
 *      sample implementation of hw on XUI with UIL
 *      to compile this, enter;
 *              % cc hw_uil.c -o hw_uil -ldwt -lX11
 *
 * Tat001       18-Jul-1989     for x11pg
 */

#include <X11/DwtAppl.h>

void hw_callback();

main (argc, argv)
int     argc;
char    *argv[];
{
        Widget  toplevel, main_window;
        DRMHierarchy    drm_id;
        static DRMRegisterArg   drm_names[] = {
                {"hw_callback", (caddr_t)hw_callback}};
        static String   uids[] = {"hw_uil.uid"};
        int     stat;

        /* initialize toolkit and DRM */
        DwtInitializeDRM ();
        stat = DwtRegisterDRMNames (drm_names, XtNumber(drm_names));
        toplevel = XtInitialize ("hw with UIL", "Test", 0, 0, &argc, argv);
        stat = DwtOpenHierarchy (XtNumber(uids), uids, 0, &drm_id);

        /* create widgets */
        stat = DwtFetchInterfaceModule (drm_id, "HW_UIL",
                        toplevel, &main_window);
        XtManageChild (main_window);

        XtRealizeWidget (toplevel);

        XtMainLoop ();
}

/* push button callback procedure */
void hw_callback (widget, tag, reason)
Widget  widget;
int     tag;
DwtAnyCallbackStruct    *reason;
{
        exit ();
}
```

```
/* end of hw_uil.c */
```

次にこのプログラムと共に用いる UIL を示します。

```
/*
 * xui_uil.uil
 *      uil definition for hw_uil.c
 *      to compile this, enter;
 *              % dxuil hw_uil.uil -o hw_uil.uid -I/usr/include/X11
 *
 * Tat001       18-Jul-1989     for x11pg
 */

module hw_uil
        names = case_sensitive
        include file 'DwtAppl.uil';
        procedure hw_callback;

        /* main_window to contain push button */
        object main_window : main_window {
                arguments {
                        width = 0;
                        height = 0;
                        main_work_window = push_button hw;
                };
                controls {
                        push_button hw;
                };
        };

        /* push button of hw */
        object hw : push_button {
                arguments {
                        label_label = "Hello world!";
                };
                callbacks {
                        activate = procedure hw_callback;
                };
        };

end module;

/* end of hw_uil.uil */
```