acconv : Athena Widget / Code Converter.

X Toolkit / Athena Widget アプリケーション開発ツール

目次

HTML 版最終修正： 1998/08/03
OpenOffice.org 版最終修正： 2005/09/30

丸市 展之 <HFD03621@nifty.ne.jp>

1 概要

acconv は、Athena Widget のアプリケーション製作の手助けをするためのツールです。
Athena Widget を始めとした、X Toolkit を使ったアプリケーションは、C 言語とオブジェクト指向ライブラリから構成されています。これにより、アプリケーションを部品ごとに管理することが出来ます。
しかし、実際のソースコードでは、部品全体の構成を把握するのは難しく、全体の構成を組み立てるのは面倒です。acconv では、このようなアプリケーション全体を実際のツリー構造で構成できるようにし、このわかりにくい作業を視覚化することを目的としています。
一般にこの種のアプリケーションは、もりだくさんな機能が盛り込まれて、かつ複雑な部品を構成するために、巨大なアプリケーションとなりますが、acconv は、Athena Widget をターゲットにしたことと、あまり大きなことを目指していないために、比較的小さくなっています。

acconv は、Athena Widget を C 言語の Code へ変換する。意味から命名しました。
以前から、X Window System に標準で収録されている Athena Widget の開発ツールです。今でも、Athena Widget ベースのツールは使われていますが、最近開発されるもので Athena Widget をターゲットにと言うのは、あまり無いのではないかと思います。

今回のリリースの acconv-0.1.3 は、0.1.2 とのバージョン番号の違いからもわかると思いますが、ソースへの修正はパッチレベルの以下の 2 箇所のみです。
(1) ツリー構造ファイル読み込みでリソースファイルが存在しないときに、エラーで異常終了する。
(2) Form ウィジェットを作成しようとしたとき、エラーで終了する。
　（Form ウィジェットのテスト部品を作成するときの条件を変更）

**(1)については、fopen() できていないのに、fclose()使用としていたところで異常終了していました。
以前は大丈夫だったのだと思うので、おそらく、関数の実装が変わったのだと思います。
(2)この変更の意味が知りたい方は、このドキュメントの 28 ページを参照してください。**

ソース以外への変更は以下の通り
(1)アプリケーションデフォルトファイルのインストール先を現在の標準的な LANG=ja_JP.eucJP の環境に合わせた。
(2)Imakefile の単純化。（必要ないと判断した部分を削除）
(3)OpenOffice.org 版(PDF 版)ドキュメントの添付。

OpenOffice.org 版ドキュメントと OpenOffice.org 版ドキュメントの PDF エクスポート版は、acconv-0.1.3 で始めて用意しました。こちらには、(project 配下の)開発系ドキュメントも統合
（0.1.2 の頃には、OpenOffice.org は、存在していませんし…）

なお、HTML 版ドキュメントは、スタイルぐらいしか変更していません。こちらは、大幅変更しました。

2 コンパイル方法

以下の手順でコンパイルします。

```
$ xmkmf
$ make depend 省略可
$ make
```

インストールする場合

```
$ su
# make install
```

オンラインマニュアルは、用意していません。

個人でのみ使用する場合で、インストールしない場合

```
cp -p Acconv.ujis.ad ~/Acconv
```

3 基本オペレーション

acconv の基本的な操作方法を示します。

3.1 メインウィンドウ

2 つのメニューと部品ツリーとその操作部分。基本操作パネル（スクロールバー, 部品選択リスト, オペレーションボタン）からなります。

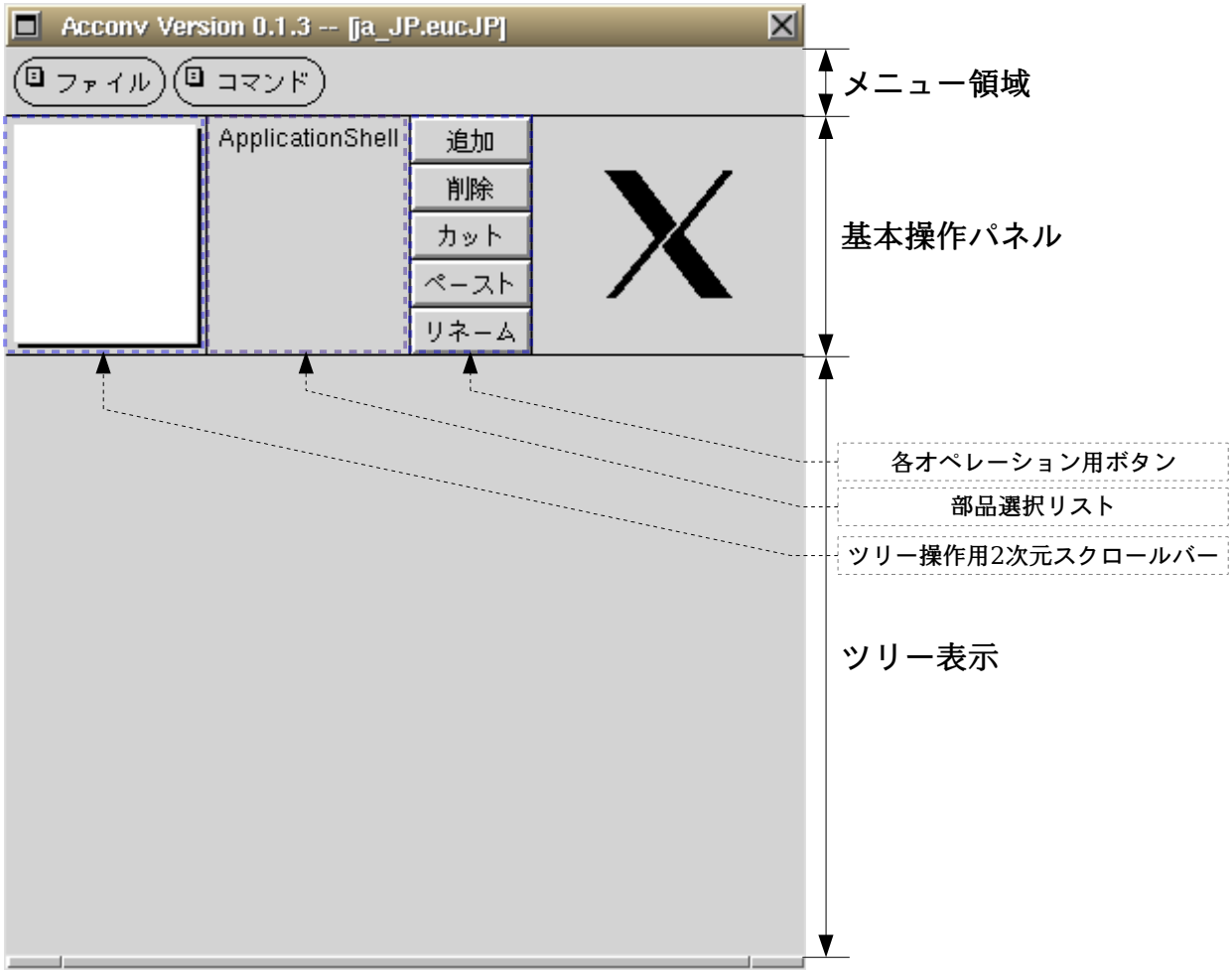


図 *1* メインウィンドウ

3.2 基本機能

acconv は、画面を構成する部品（ウィジェット）のツリー構造としての配置を決定していくことで、Athena Widget を使ったアプリケーションの構成を決めるツールです。

このツリー構造作成のためには、基本操作パネルを操作してツリー構造を編集します。この操作を行う各オペレーションボタンの機能を示します。

表 1 基本操作パネルのオペレーションボタン

日本語リソース	英語版リソース	説明
追加	**Add**	部品の追加を行ないます。 使い方は、次の章で
削除	**Delete**	部品を削除します。
カット	**Cut**	使えません。（まだ、出来てません）
ペースト	**Paste**	使えません。（まだ、出来てません）
リネーム	**Rename**	ウィジェット名、および変数名を変更できます。

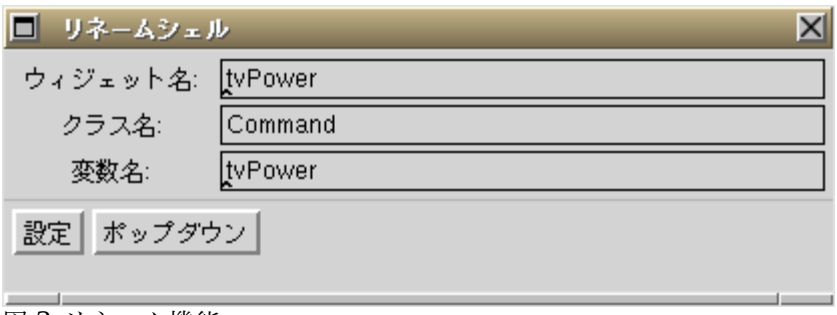
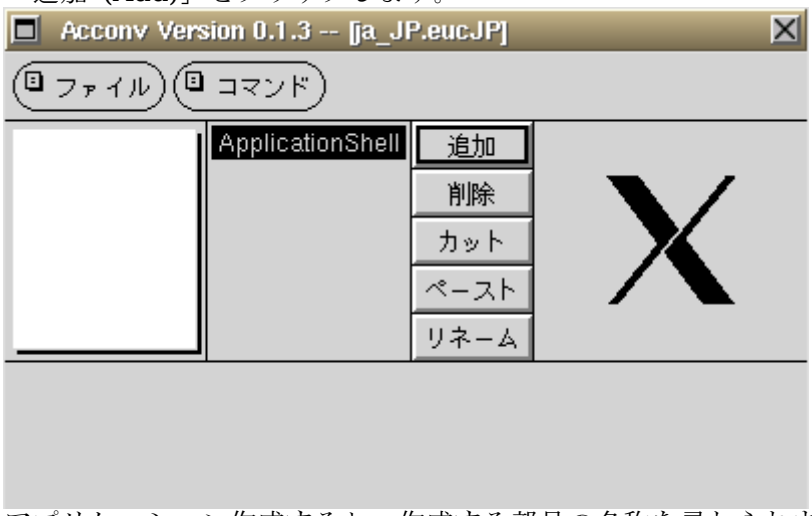


図 2 リネーム機能

3.3 部品の作成

acconv でアプリケーションを作成する場合、最初の部品は、ApplicationShell になります。

(1)作成する方法は、部品選択リストから ApplicationShell （しかないが）選択を選択して、「」「追加 (Add)」をクリックします。



アプリケーション作成すると、作成する部品の名称を尋ねられます。

部品の名称を入力して「設定&ポップダウン」(Set & Popdown)を押して部品を作成します。

(2)この後は、通常の部品を作成していきます。２個目以降の部品は、 どの部品の子供として作成するかを決めます。次の部品を作成選択する部品を選ぶと、使用可能な部品がリストに表示されます。 この後は、最初の部品と同じように、部品を作成して行きます。

3.4 メニュー

acconv では、ファイルメニュー / コマンドメニュー の 2 つのメニューから 構成されています。

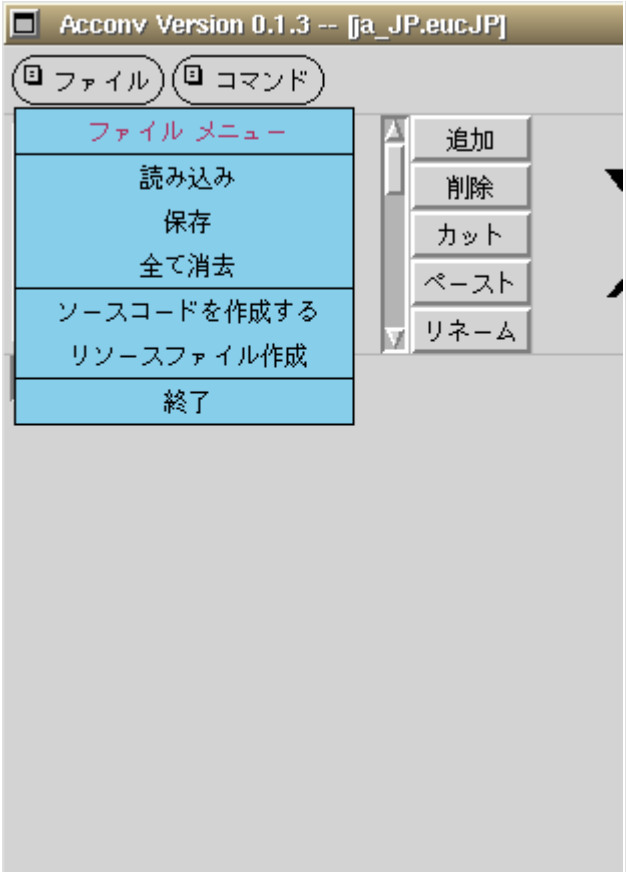


図 *3* ファイルメニュー

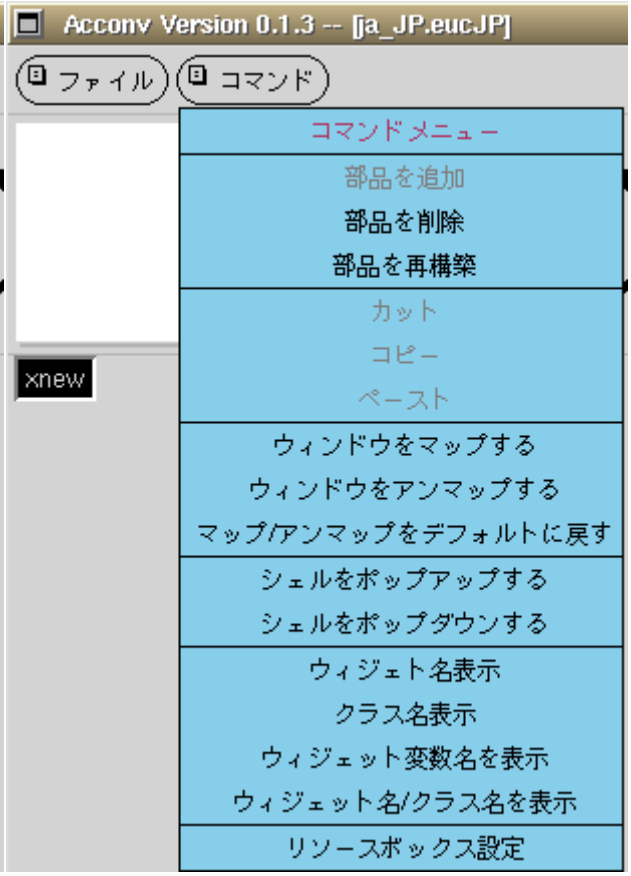


図 *4* コマンドメニュー

表 2 ファイルメニュー

日本語リソース	英語版リソース	説明
ファイルメニュー	**File Menu**	メニューのタイトルです。
読み込み	**Load**	部品構成ファイルを読み込みます。
保存	**Save**	部品構成ファイルを書き込みます。
全て商況	**Clear All**	すべての部品を破棄して、初期状態に戻します。
ソースコードを作成する	**Make Source File**	部品を構成に従って、C 言語のソースコードを作成します。これは、acconv が 機械的に生成するものです。
リソースファイル作成	**Make Resource File**	定義済みのリソースと新たに定義したリソースの 2 種類のリソースをテキストで表示します。
終了	**Quit**	プログラムを終了します。

表３ コマンドメニュー

Parts Add	部品の追加を行ないます。
Parts Delete	部品を削除します。動作が不安定なので、メインウィンドウのボタンを使って下さい。
Cut	使えません。（まだ、出来てません）
Copy	使えません。（まだ、出来てません）
Paste	使えません。（まだ、出来てません）
Map	部品をマップします。
Unmap	部品をアンマップします。
(Un)map Default	部品をマップ／アンマップの状態を初期化します。
Shell-parts Popup	ポップアップシェル／メニューをポップアップします。
Shell-parts Popdown	ポップアップシェル／メニューをポップダウンします。
Show Widget Names	ウィジェット名を表示します。
Show Class Names	クラス名を表示します。
Show Widget Variable	ウィジェット+変数名を表示します。
Show Widget & Class Name	ウィジェット名+クラス名を表示します。
Show Resource Box	リソース設定ウィンドウをポップアップします。

3.5 作成したウインドウ／ポップアップシェル／メニューのテスト表示

作成したウインドウ／ポップアップシェルは、Map 機能や、Shell-parts Popup 機能でテスト表示することが可能です。以下は、実際のアプリケーションと テスト表示結果のサンプルです。

図 *5 xwireless* の本物の画面 *(Xaw3D*版*)*

図 *6 acconv* のテスト表示 *(*リソースファイルの例*)*

図 *7 acconv* のテスト表示 *(*リソース無しの例*)*

図 5 が実際のアプリケーションで、図 6 が acconv によるテスト表示です。テスト表示も実際のアプリケーションのようにリソースが設定できていないと図 7 の様に期待した表示とは、異なることになります。
実際には、アプリケーション内部で設定されるリソースや acconv アプリケーション自体のリソースが悪影響を与えることなどの要因で、「実際のアプリケーション」と「acconv のテスト表示」は、全然似ないこともあります。
図 5 と図 6 の違いで選択されているフォントは、個人的なリソース設定、メニューボタンの形状は、Acconv アプリケーションデフォルトファイルの中の設定に依存しています。各ボタンのビットマップは、実際のアプリケーションでは内部で配置しているためです。

リソースは、プログラムに埋め込むことも可能なためアプリケーションデフォルトファイルの形式になっていないものや、アプリケーションデフォルトファイルで設定されるリソースが一部のみで設定が不十分のこともあります。

4 acconv の応用

NEW
acconv には、「実際の画面上でのテスト機能」,「リソース作成機能」,「ソース作成」などの機能があります。
上記のような機能をうまく使えば、acconv で部品構成を考えて、最終的なアプリケーション上のデザインのテストし、サンプルソースを生成するためには、いくつかの知識が必要です。
実は、部品構成のみを作っただけでは、通常、実際の画面上は期待した部品配置になりません。

以下に、関連する知識を示します。
- X Toolkit と Athena Widget の知識。
- リソースの知識。
- C 言語や X Toolkit でアプリケーションをコンパイルに関する知識。

これらの知識は、サンプルデータから読み込むことで得られることもありますが、実際に動いているプログラムから得ることもできます。

以下は、実用的な使い方についてまとめます。

4.1 部品構成ファイルと *editres* との連係

acconv は、作成済みのアプリケーションから editres のように、 ツリー情報を取得は出来ないのですが、editres では、ツリー情報を commands のメニューの [Dump Widget Tree to a File] を使うと ファイルに書き出すことができます。このファイルは、acconv の 部品構成ファイルとして読み込むことができます。 このようにして作成した editres との連係の例は、前の章にあります。

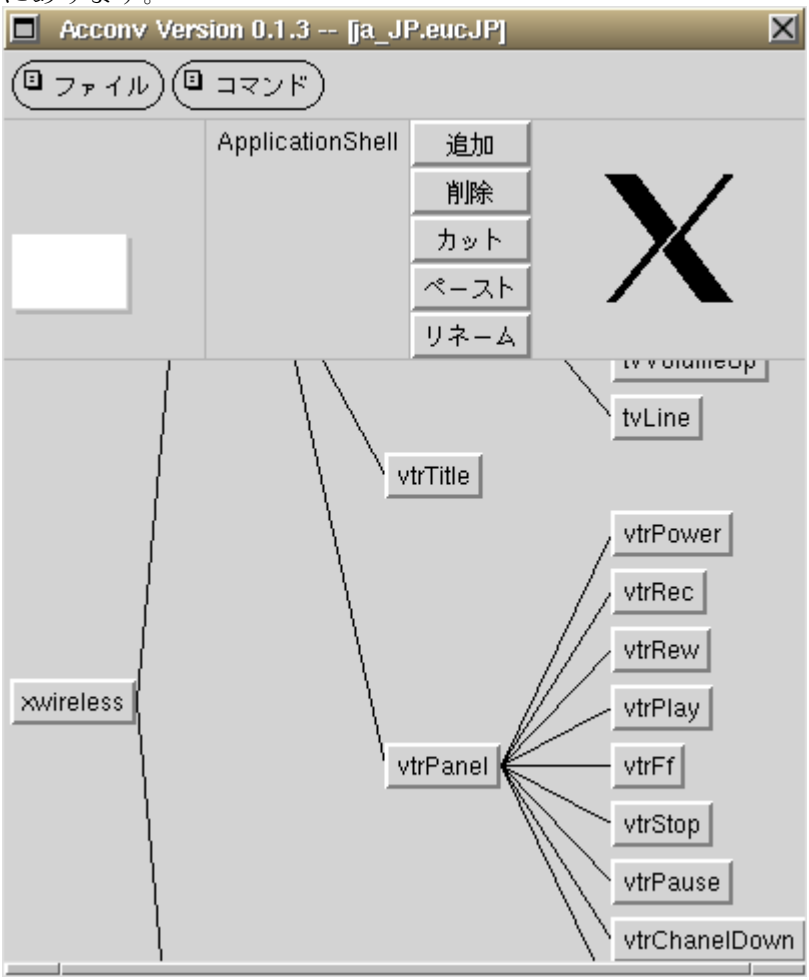


図 *8 editres* との連係例

acconv の部品構成ファイルは、このファイルの上位互換になるように設計してあります。
ただし、100％大丈夫と言うわけではなく、Text ウィジェットのように複雑なものもあるため多少、問題があるかと思います。
リソースエディタ（editres）は、既存の X Toolkit ベースのアプリケーションに対して使用することができますが、acconv で利用できるものは、標準の Athena Widget のみで構成されたアプリケーションのみになります。つまりアプリケーション独自のウィジェットが含まれているもの xclock, xterm/kterm なども、再現する事ができない部品を含んでいることになります。子のようなアプリケーションも相当数あります。

なお editres の機能は、実際に動作しているアプリケーションの部品構成を取得して設定を動的に編集できるような非常に強力なものです。このツールを使うことで、X のリソースに関して多くの知識を得ることができると思います。

そのほか、フォントに関する情報は xfontsel, xfd, xlsfonts などのコマンドも有効です。
acconv で扱う部品は、X Toolkit と Athena Wdiget のウィジェット[1]と呼ばれるオブジェクトです。X Toolkit は C 言語で書かれているため C++のオブジェクトとは、異なり C 言語の構造体で作られたものです。実態は構造体ですが、継承など考え方もあります。
オブジェクト的な継承などの関係は、viewres と言うアプリケーションで見ることができます。リソースの一覧も見ることができます。

1 ウィジェットだけでなく一部ガジェットとよばれるものもも含みます。

4.2 ソースコードの生成

部品を構成するとファイルメニューの Make Source File を選択 することで、C 言語のソースコードができます。これは、acconv が機械的に生成したものなので、かなり汚いです。これをこのまま使いたい方もいらっしゃるかも知れませんが、基本的には、コピー&ペーストしてソースコードに張り付けて使用するのがいいと思います。
右のフロッピーディスクのアイコンをクリックするとファイル選択ダイアログが表示されます。ここで選択するか、直接テキストを入力することでファイルの指定して保存するができます。
なお、「読込」ボタンについては、動作しません。

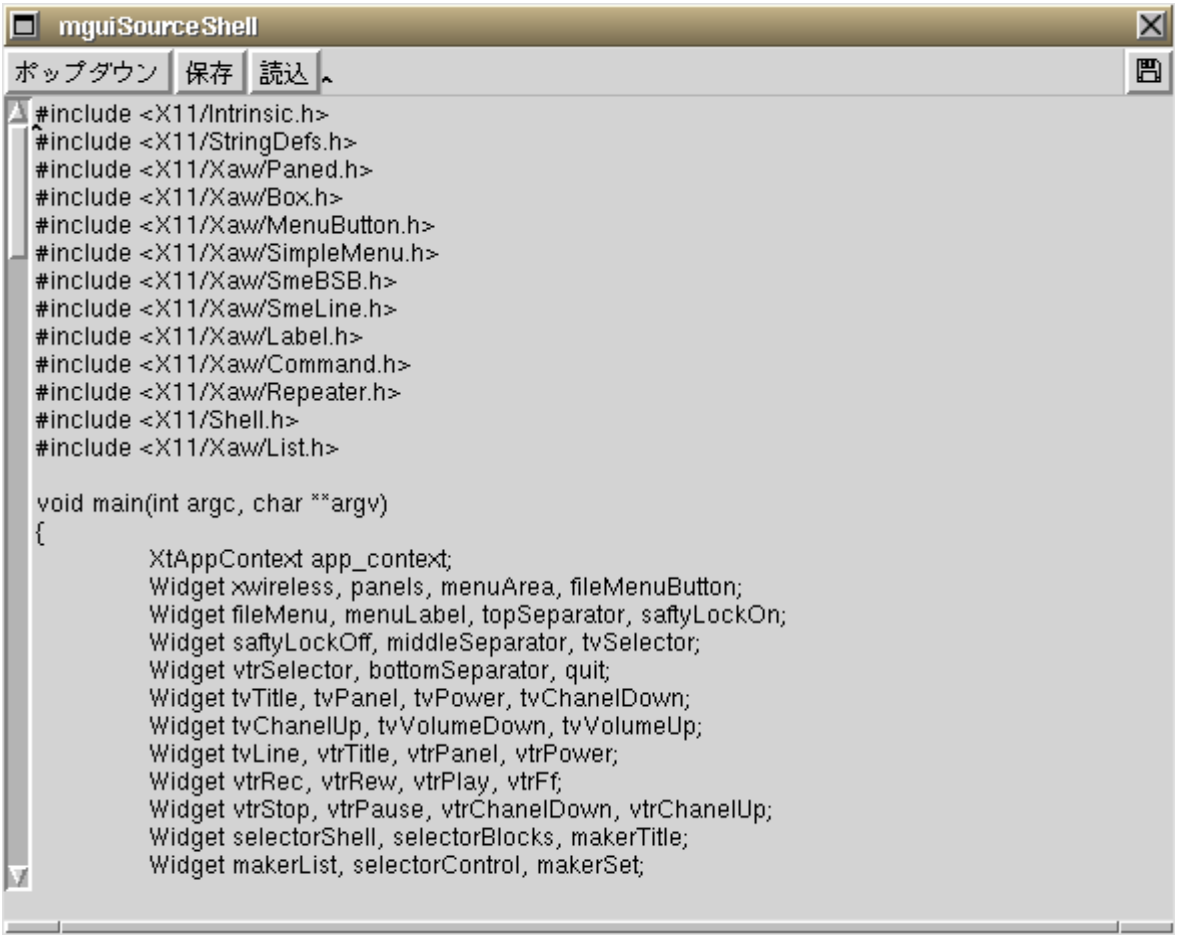


図 9 サンプルソースコードのシェル

4.2.1 サンプルソースのコンパイル

作成したソースコードは、以下のようにしてコンパイルすることができます。

```
digit@pcgv505:~$ cc -L /usr/X11R6/lib -lXt -lXaw xwireless.c -o xwireless
xwireless.c: 関数 `main' 内:
xwireless.c:14: 警告: `main' の戻り値の型が `int' ではありません
```

サンプルソースコードのコンパイルの例

実際にこのプログラムを実行すると以下の図 10 ようになります。
（この例は、リソースファイルを用いています）

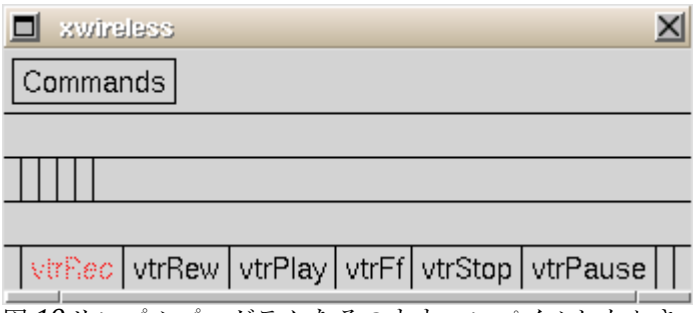


図 *10* サンプルプログラムをそのままコンパイルしたとき

これでは、「いままで紹介した本物（8 ページの図 5）や acconv のサンプル（図 6）ともずいぶん異なるのでは?」と思われたかと思います。
これは、元の xwireless が日本語化されているのにサンプルソースコードが、国際化[*1]されていないためです。
国際化するためには、以下のように XtSetLanguageProc() を追加する。

```
digit@pcgv505:~$ diff -u xwireless.c.orig xwireless.c | expand -4
--- xwireless.c.orig    2005-09-23 11:32:30.000000000 +0900
+++ xwireless.c 2005-09-23 11:47:35.000000000 +0900
@@ -21,6 +21,9 @@
    Widget tvLine, vtrTitle, vtrPanel, vtrPower;
    Widget vtrRec, vtrRew, vtrPlay, vtrFf;
    Widget vtrStop, vtrPause, vtrChanelDown, vtrChanelUp;
+
+    XtSetLanguageProc(NULL, NULL, NULL);
+
    xwireless = XtVaAppInitialize(
              &app_context,
              "Xwireless",
```

サンプルプログラムを国際化のするために必要な修正

この修正を追加すると以下のようになります。

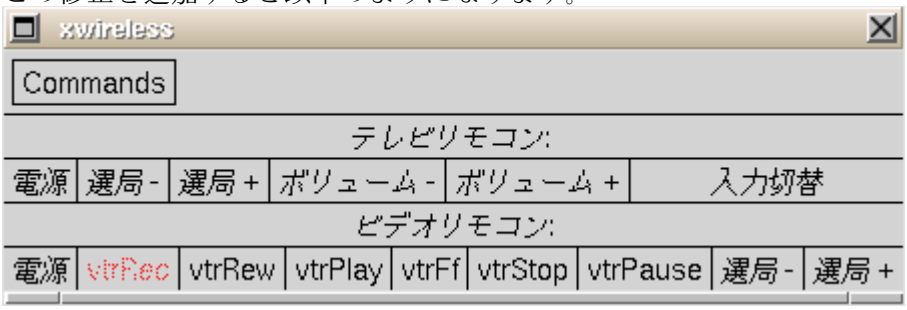


図 *11* サンプルプログラムを国際化したときの例

*1 多くの言語へ対応可能になるようにプログラムが組まれていること。（実際に特定の言語が使えるかどうかとは、関係がなく、特定の言語圏へ対応させることは地域化となります）

4.3 リソースの設定

8 ページの図 5～図 7 を参照していただくといいと思いますが、これまでの説明で、リソース設定によりプログラムの概観が大きく変わることもあることは。acconv は、ほとんど本当のウィジェットを使ってテスト表示を行うためリソースの読み込み機能と追加設定機能を持ちます。

設定済みのリソースや追加設定したリソースを削除する機能が無いため、リソース管理の機能は不十分ではあるのですが…。

4.3.1 追加リソースの設定画面

リソース編集機能 部品を選択しコマンドメニューの Show Resource Box を選択する ことで、リソース変更ウィンドウがでます。これは、editres の リソースボックスと同様にリソースの変更を可能にしています。

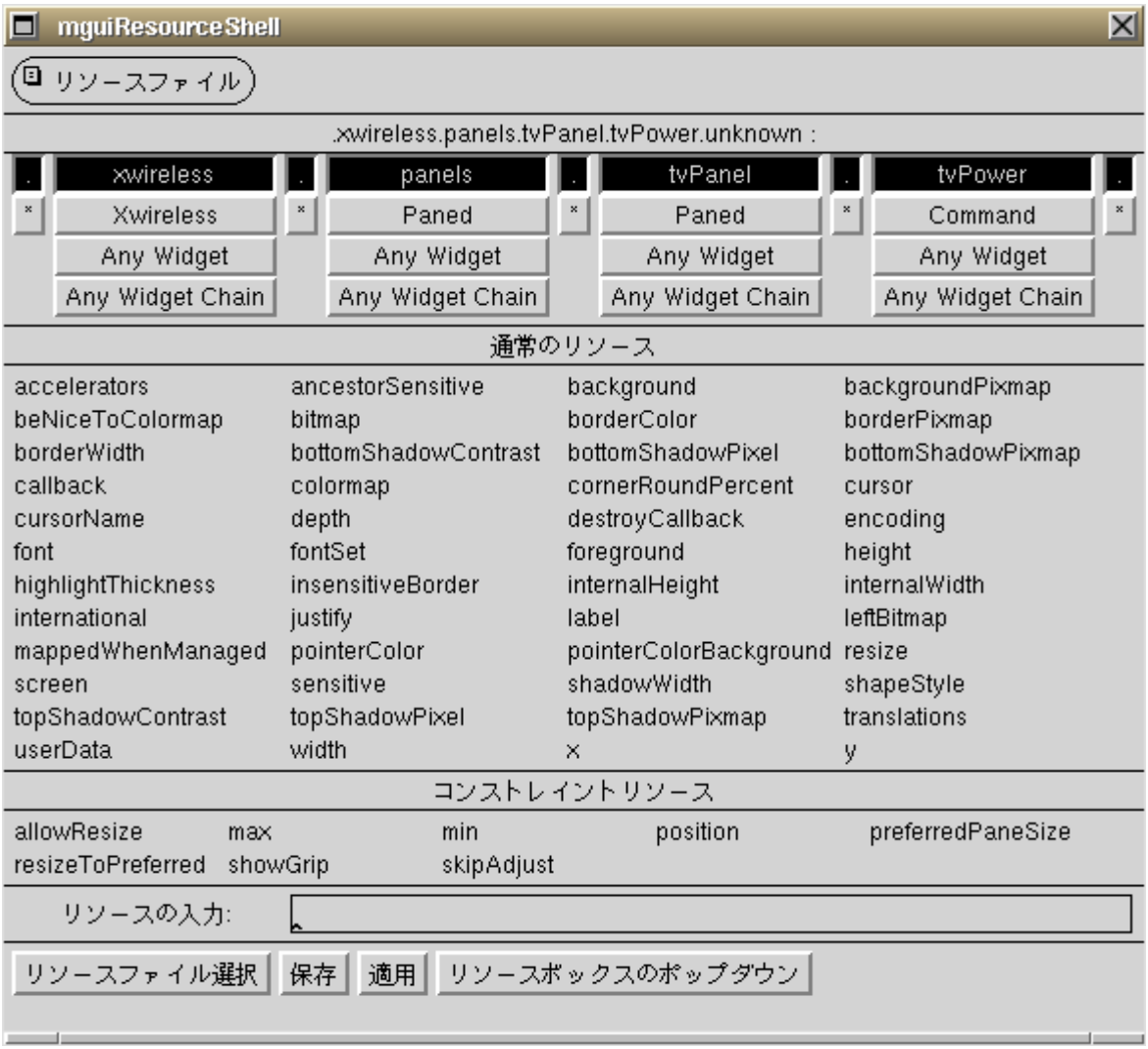


図 *12* リソースの設定画面

ただし、リソースファイル選択（Select resorce file）/ 保存（Save）は、無効です。

acconv の現在の仕様でテストウィジェットにリソースを反映させるには、先頭の文字を疎結合を示す「*」にする必要があります。
これはテストウィジェットの場合に acconv（クラス名：Acconv）の配下にテストウィジェットを作るために密結合にするとマッチングしません。

4.3.2 設定されているリソースの一覧画面

リソースデータベース表示機能 ファイルメニューの Make Resource File を選択すると、このアプリケーション に関するリソースを定義済みのリソースと新たに定義したリソースの 2 種類のリソースをテキストで示します。

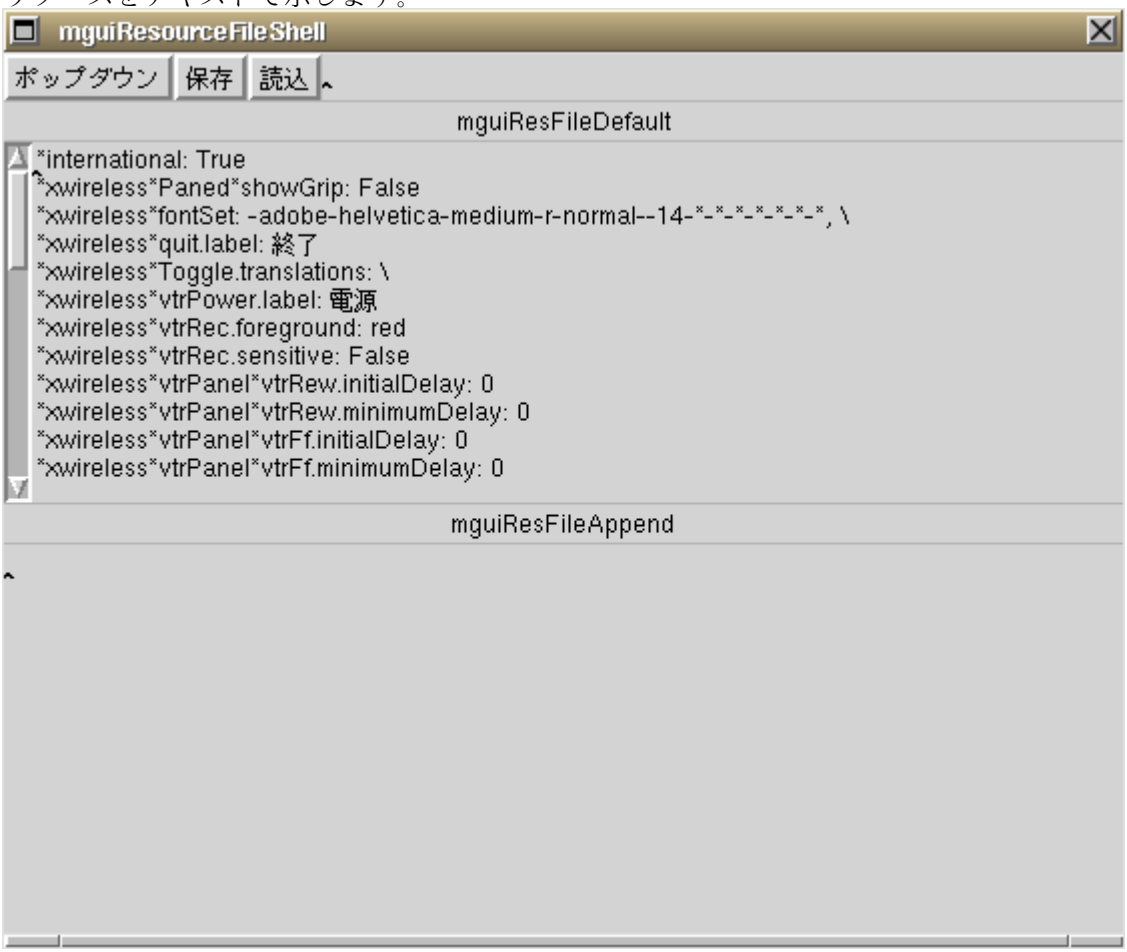


図 *13* リソースファイルと追加リソースのシェル

ただし、Load / Save は、無効です。

5 acconv の設計

NEW

ここでは、Athena Widget ベースのアプリケーションのプログラムサンプルを生成するためのプログラムである acconv の内部のことと Athena Widget に付いて補足します。
なお、このプログラムの設計に関しては、C 言語の構造体、ポインタ、関数へのポインタ、可変引数の関数についての基本的な知識があることが前提となっています。

5.1 Athena Widget

X Toolkit のサンプルとも言える、基本的な GUI パーツ集。2 次元的な形状を持つ。3 次元的な概観を持つ拡張ライブラリに置き換えることで、デザインを変更することもできます。

5.2 ツリー構造とデータ構造

以下は、acconv で作成したツリー構造データがどのようなデータから構成されるかを示す概略図です。

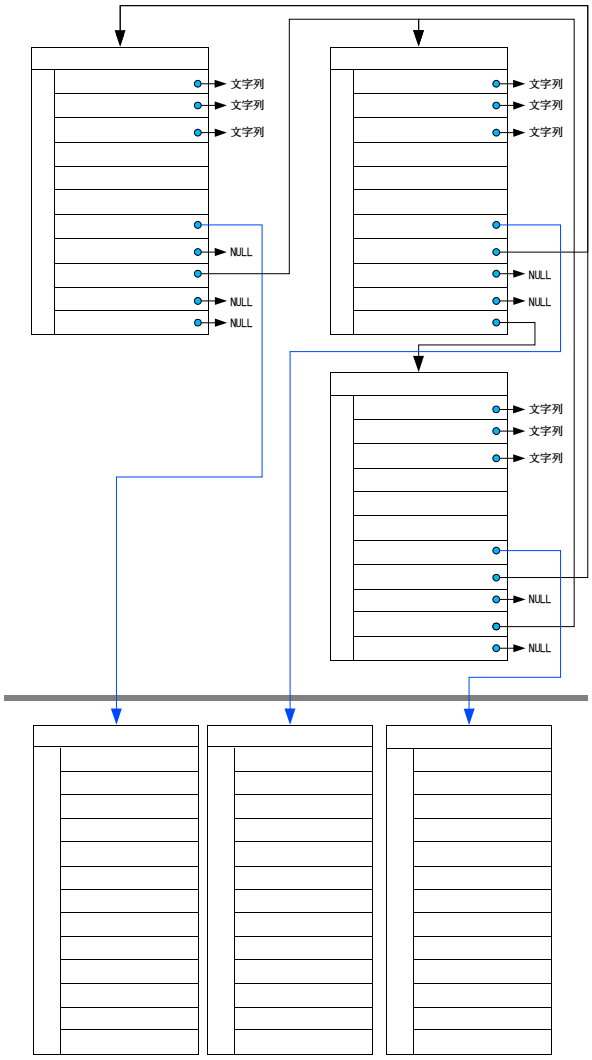


図 *14* ツリー構造のデータ構成

まず、各部品を構成する PartsTree 型データが、部品毎に存在します。つまり左記の図は、3 つの部品から構成されたツリー構造を示します。

ツリー構造は、parent_ptr, child_ptr, prev_ptr, next_ptr, のメンバー変数で示されています。この親子関係は、オブジェクトとしての継承とは関係ありません。

athenaInfo で部品種別毎のデータ情報を示すメンバー変数もあります。athenaInfo が示す部品情報は、AWidgetInfo 型のデータでこれは、（Label ウィジェットや Command ウィジェットなどの）部品の種類分あります。

5.2.1 PartsTree 型データ（部品ツリー構造情報の構造体）について

以下が、PartsTree 型（部品ツリー構造情報の構造体）宣言部分です。

```
typedef struct _PartsTree {
   char *widgetName;
   char *variableName;
   char *className;
   Widget partsWidget;
   Widget treeWidget;
   int mapping;
   struct _AWidgetInfo *athenaInfo;
   struct _PartsTree *parent_ptr;
   struct _PartsTree *child_ptr;
   struct _PartsTree *prev_ptr;
   struct _PartsTree *next_ptr;
} PartsTree;
```

PartsTree 型（部品ツリー構造情報の構造体）宣言

この構造体は、作成するアプリケーションの部品となるウィジェット（Widget）毎に一つずつ作成されます。上記の宣言に含まれるメンバーの説明を以下の表 4 に示します。

表 4 PartsTree 型：部品ツリー構造情報

メンバー変数	説明
widgetName	ウィジェット名
variableName	ウィジェットの変数名
className	クラス名（クラス・ポインタ名）
partsWidget	テスト実現化用のウィジェット
treeWidget	ツリー表示用のウィジェット
mapping	テスト的なマッピングの有無（真：マッピングする / 偽：しない）
athenaInfo	部品種類総合情報（AWidgetInfo 型）
parent_ptr	部品構成ツリーで親を指すポインタ
child_ptr	部品構成ツリーで子を指すポインタ
prev_ptr	部品構成ツリーで兄（同じ親の直前）を指すポインタ
next_ptr	部品構成ツリーで弟（同じ親の直後）を指すポインタ

この構造体は、以下の部分からなります。

- 名前を示す文字列のポインタ。（widgetName, variableName, className）
- ウィジェットの情報やテスト機能のための部分。
  （partsWidget, treeWidget, mapping, athenaInfo）
- ツリー構造を構成する自己参照用のポインタ。
  （parent_ptr, child_ptr, prev_ptr, next_ptr）

5.2.2 AWidgetInfo 型データ（部品種類総合情報の構造体）について

以下が、AWidgetInfo 型（部品種類総合情報の構造体）宣言部分です。

```
typedef struct _AWidgetInfo {
    const char *publicInclude;
    const char *className;
    const char *variableName;
    WidgetClass *partsClass;
    const char *instance;
    int *argTypes;
    int childExist;
    int type;
    Boolean (*remakable)(struct _AWidgetInfo *myWidget, PartsTree *parts);
    Boolean (*creatable)(struct _AWidgetInfo *myWidget, PartsTree *parts);
    Boolean (*realizable)(struct _AWidgetInfo *myWidget, PartsTree *parts);
    void (*widgetCreation)(struct _AWidgetInfo *myWidget, PartsTree *parts);
    int id;
} AWidgetInfo;
```

AWidgetInfo 型（部品種類総合情報の構造体）宣言

この構造体は、作成するアプリケーションの部品となるウィジェット（Widget）の種類毎に一つずつ作成されます。

AwidgetInfo 型のデータで部品の種類情報を宣言している widgetSet[]配列の一部を抜粋します。この widgetSet[]配列のデータ構造を図 15 に、部品種類の一覧を表 6 に示します。

*widgetSet[]*配列（*AWidgetInfo* 型）の初期化宣言

```
AWidgetInfo widgetSet[] = {
   {
      "#include <X11/Composite.h>",   /* インクルード・ファイル */
      "Composite",                    /* クラス名 */
      "compositeWidgetClass",         /* 変数名（クラス・ポインタ） */
      &compositeWidgetClass,          /* クラス変数（クラス・ポインタ） */
      NULL,                           /* インスタンス化フォーマット文字列 */
      NULL,                           /* インスタンス化引数情報 */
      NoChild,                        /* 子ウィジェットの実現条件 */
      MetaClass,                      /* 部品種類 */
      NULL,                           /* 再構成可能判定関数 */
      NULL,                           /* ツリー構造作成可能判定関数 */
      NULL,                           /* ウィジェット実現可能判定関数 */
      NULL                            /* 部品作成関数 */
   }, {
      "#include <X11/Xaw/Box.h>",     /* インクルード・ファイル */
      "Box",                          /* クラス名 */
      "boxWidgetClass",               /* 変数名（クラス・ポインタ） */
      &boxWidgetClass,                /* クラス変数（クラス・ポインタ） */
      createNormal,                   /* インスタンス化フォーマット文字列 */
      createNormalArgs,               /* インスタンス化引数情報 */
      MultiChild,                     /* 子ウィジェットの実現条件 */
      NormalWidget,                   /* 部品種類 */
      hasFamily,                      /* 再構成可能判定関数 */
      partsExist,                     /* ツリー構造作成可能判定関数 */
      NULL,                           /* ウィジェット実現可能判定関数 */
      widgetCreater                   /* 部品作成関数 */
   }, {
       .
       .
       .
   }, {
      "#include <X11/Xaw/SmeLine.h>", /* インクルード・ファイル */
      "SmeLine",                      /* クラス名 */
      "smeLineObjectClass",           /* 変数名（クラス・ポインタ） */
      &smeLineObjectClass,            /* クラス変数（クラス・ポインタ） */
      createNormal,                   /* インスタンス化フォーマット文字列 */
      createNormalArgs,               /* インスタンス化引数情報 */
      NoChild,                        /* 子ウィジェットの実現条件 */
      SmeObject,                      /* 部品種類 */
      hasParent,                      /* 再構成可能判定関数 */
      partsExist,                     /* ツリー構造作成可能判定関数 */
      NULL,                           /* ウィジェット実現可能判定関数 */
      widgetCreater                   /* 部品作成関数 */
   }
};
```

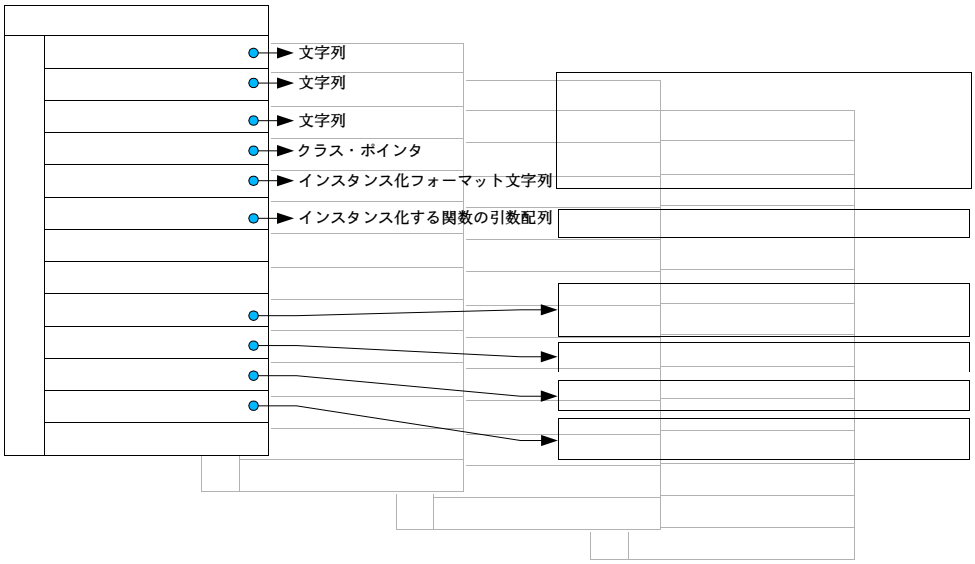


図 *15 widgetSet[]* 配列のデータ構成

上記の宣言に含まれるメンバーの説明を以下の表 5 に示します。

表 5 AWidgetInfo 型：部品種類総合情報

メンバー変数	S	W	T	説明
publicInclude	●			パブリック・インクルード・ファイル定義 - 文字列 ソースファイルを作成するときに取り込むヘッダーファイルを呼び出すための宣言文です。
className	●		●	クラス名 - 文字列
variableName	●			変数名（クラス・ポインタ） - 文字列
partsClass		●		クラス・ポインタ
instance	●			インスタンス化 - フォーマット文字列
argTypes	●			引数種別配列
childExist			●	子ウィジェットの実現条件 (NoChild:0/OneChild:1/MultiChild:EOF)
type			●	部品種類(NoWidget:-1/MetaClass:0x8000/NormalWidget:0x0001/ ShellWidget:0x0010/SimpleMenu:0x0030/SmeObject:0x0100)
remakable			●	再構成可能判定関数（NULL, hasChild, hasFamily, hasParent）
creatable			●	ツリー構造作成可能判定関数（NULL, hasNoFamily, partsExist）
realizable		—	—	テスト部品実現化可能判定関数（NULL, hasNormalChild）
widgetCreation		●		部品作成関数（NULL, shellCreater, widgetCreater）
id	—	—	—	未使用

S:ソースコード作成用の情報 / W:テストウィジェット作成用の情報 / T:ツリー構造管理用の情報

この構造体は、以下の部分からなります。

- 文字列のポインタ。（publicInclude, className, variableName）
- ソースコードを生成するための情報。（instance, argTypes）
- 部品の情報。（childExist, type, id）
- ツリー構造を構成などの管理に使用する関数を示すポインタ。
  （remakable, creatable, realizable, widgetCreation）

```
Boolean hasParent(AWidgetInfo *myWidget, PartsTree *parts);
Boolean hasChild(AWidgetInfo *myWidget, PartsTree *parts);
Boolean hasFamily(AWidgetInfo *myWidget, PartsTree *parts);
Boolean hasNoFamily(AWidgetInfo *myWidget, PartsTree *parts);
Boolean hasNormalChild(AWidgetInfo *myWidget, PartsTree *parts);
Boolean partsExist(AWidgetInfo *myWidget, PartsTree *parts);
void widgetCreater(AWidgetInfo *myWidget, PartsTree *parts);
void shellCreater(AWidgetInfo *myWidget, PartsTree *parts);
```

(1) publicInclude メンバー（const char 型のポインタ：文字列）
ソースコード上で、ヘッダーファイルの読み込みを行う、以下のような文字列が格納される

```
    "#include <X11/Xaw/Box.h>",      /* インクルード・ファイル */
```

(2) className メンバー（const char 型のポインタ：文字列）
ソースコード上で、クラス名を指定するところやツリー構造管理などで使用する。

```
    "Composite",                     /* クラス名 */
```

(3) variableName メンバー（const char 型のポインタ：文字列）
ソースコード上で、クラス変数名を指定するところで使用する。

```
    "compositeWidgetClass",          /* 変数名（クラス・ポインタ） */
```

(4) partsClass メンバー（WidgetClass 型のポインタ）
テスト用のウィジェットの生成とリソース設定のために使用する。

```
    &compositeWidgetClass,           /* クラス変数（クラス・ポインタ） */
```

(5) instance メンバー（const char 型のポインタ：文字列）
ここには、文字列に相当する const char 型のポインタが格納されます。

```
    createNormal,                    /* インスタンス化フォーマット文字列 */
```

このポインタが示す文字列を示します。
以下が、instance（インスタンス化 - フォーマット文字列）に割り当てされる文字列の例。

```
char createNormal[] =
    "\t%s = XtVaCreateManagedWidget(\n"
    "\t\t\t\t\"%s\",\n"
    "\t\t\t\t%s,\n"
    "\t\t\t\t%s,\n"
    "\t\t\t\tNULL);\n";
```

別に文字列を宣言しているのは、同じ様な宣言を用いることが多いからです。

(6) argTypes メンバー（int 型へのポインタ）
ここには、整数配列の先頭を指すポインタです。以下のように初期化されます。

```
    createNormalArgs,                /* インスタンス化引数情報 */
```

argTypres（引数種別配列）に割り当てられたポインタは、以下の様に別に宣言されて割り当てられている配列です。このポインタの指す値は、上記のフォーマット文字列に対応した引数を記録した引数を示す情報です。

```
int createNormalArgs[] =
    {ArgWidget, ArgWidgetName, ArgWidgetClass, ArgParent, NoArg};
```

instance と argTypes の二つの情報からウィジェットを生成するソースコードを作ることができます。

(7) childExist メンバー（int 型）
この childExist メンバーは、この種の Widget が子ウィジェットをどのような持つ可能性があるのかを示します。

```
    MultiChild,                      /* 子ウィジェットの実現条件 */
```

この childExist メンバーは、以下の値を格納しています。

```
#define NoChild     (0)
#define OneChild    (1)
#define MultiChild  (EOF)
```

ラベル	整数値	意味
NoChild	0	子ウィジェットを配置できない
OneChild	1	子ウィジェットは、1 つのみ
MultiChild	EOF	複数のウィジェットを配置できる。

この値は、現時点で使用していません。将来のツリー構造 - 判定材料として検討中の値となります。

(8) type メンバー（int 型）

この type メンバーは、部品となるオブジェクトの種類を分類しています。この分類は、acconv 独自の管理上の問題で分類しているだけです。

ラベル	整数値		意味
NoWidget	-1		予約済み
MetaClass	0x8000		
NormalWidget	0x0001		
ShellWidget	0x0010		
SimpleMenu	0x0030		
SmeObject	0x0100		

(9) remakable メンバー（関数へのポインタ）

この remakable メンバーは、再構成可能判定関数を指すポインタです。
以下のように初期化します。

```
    hasFamily,                          /* 再構成可能判定関数 */
```

ここに記述されるのは、以下のようなプロトタイプ宣言に対応する関数の関数名です。

```
Boolean hasParent(AWidgetInfo *myWidget, PartsTree *parts);
Boolean hasChild(AWidgetInfo *myWidget, PartsTree *parts);
Boolean hasFamily(AWidgetInfo *myWidget, PartsTree *parts);
```

該当する関数が存在しない場合には、NULL として NULL ポインタを指定します。

(10) creatable メンバー（関数へのポインタ）

この creatable メンバーは、ツリー構造作成可能判定関数を指すポインタです。
以下のように初期化します。

```
    partsExist,                         /* ツリー構造作成可能判定関数 */
```

ここに記述されるのは、以下のようなプロトタイプ宣言に対応する関数の関数名です。

```
Boolean hasNoFamily(AWidgetInfo *myWidget, PartsTree *parts);
Boolean partsExist(AWidgetInfo *myWidget, PartsTree *parts);
```

該当する関数が存在しない場合には、NULL として NULL ポインタを指定します。

(11) realizable メンバー（関数へのポインタ）

この realizable メンバーは、テスト部品実現化可能判定関数を指すポインタです。
以下のように初期化します。

```
    partsExist,                         /* ツリー構造作成可能判定関数 */
```

ここに記述されるのは、以下のようなプロトタイプ宣言に対応する関数の関数名です。

```
Boolean hasNormalChild(AWidgetInfo *myWidget, PartsTree *parts);
```

該当する関数が存在しない場合には、NULL として NULL ポインタを指定します。

(12) widgetCreation メンバー（関数へのポインタ）

この widgetCreation メンバーは、部品作成関数を指すポインタです。
以下のように初期化します。

```
    widgetCreater                       /* 部品作成関数 */
```

ここに記述されるのは、以下のようなプロトタイプ宣言に対応する関数の関数名です。

```
void widgetCreater(AWidgetInfo *myWidget, PartsTree *parts);
void shellCreater(AWidgetInfo *myWidget, PartsTree *parts);
```

該当する関数が存在しない場合には、NULL として NULL ポインタを指定します。

(13) id メンバー（int 型）

id メンバーは、「widgetSet[]配列（AWidgetInfo 型）の初期化宣言」での初期化されていません。このメンバーは、以下の関数で初期化します。現時点では特に使用していません。

```
/*****************************************************************************
 *  widgetInit：widgetSet の識別番号(ID)を初期化する
 *  形式
 *      void widgetInit(void);
 *  説明
 *
 *****************************************************************************/
void widgetInit(void)
{
   int counter, numOfClass;

   /*********************************************
    *  パブリック・インクルードファイル部分の作成
    *********************************************/
   numOfClass = sizeof(widgetSet)/sizeof(widgetSet[0]);
   for(counter = 0;counter < numOfClass;++counter) {
      widgetSet[counter].id = counter;
   }
}
```

widgetSet[]配列の id メンバーを初期化する関数

表 6 widgetSet[] 配列と対応する部品種類

変数	対応するクラス	クラスの種類[*1]	ウィジェットの種類
widgetSet[0]	Composite	Xt	インスタンス化不可
widgetSet[1]	Box	Xaw	ウィジェット
widgetSet[2]	Constraint	Xt	インスタンス化不可
widgetSet[3]	Form	Xaw	ウィジェット
widgetSet[4]	Dialog	Xaw	ウィジェット
widgetSet[5]	Viewport	Xaw	ウィジェット
widgetSet[6]	Paned	Xaw	ウィジェット
widgetSet[7]	Tree	Xaw	ウィジェット
widgetSet[8]	Porthole	Xaw	ウィジェット
widgetSet[9]	OverrideShell	Xt	シェルウィジェット
widgetSet[10]	SimpleMenu	Xaw	シェルウィジェット
widgetSet[11]	TopLevelShell	Xt	シェルウィジェット
widgetSet[12]	ApplicationShell	Xt	アプリケーション
widgetSet[13]	TransientShell	Xt	シェルウィジェット
widgetSet[14]	Simple	Xaw	インスタンス化不可
widgetSet[15]	Grip	Xaw	ウィジェット
widgetSet[16]	Label	Xaw	ウィジェット
widgetSet[17]	Command	Xaw	ウィジェット
widgetSet[18]	MenuButton	Xaw	ウィジェット
widgetSet[19]	Repeater	Xaw	ウィジェット
widgetSet[20]	Toggle	Xaw	ウィジェット
widgetSet[21]	List	Xaw	ウィジェット
widgetSet[22]	Panner	Xaw	ウィジェット
widgetSet[23]	Scrollbar	Xaw	ウィジェット
widgetSet[24]	StripChart	Xaw	ウィジェット
widgetSet[25]	Text	Xaw	ウィジェット
widgetSet[26]	Sme	Xaw	ガジェット
widgetSet[27]	SmeBSB	Xaw	ガジェット
widgetSet[28]	SmeLine	Xaw	ガジェット

上記のクラスの派生に対する情報は、viewres を見るといいかと思います。

*1 Xaw は Athena Widget のクラスで、Xt は X Toolkit のクラス。もともと、X Toolkit のクラスが派生元になっているが、X Toolkit のクラスだけでは単純には使用できないものも多い。

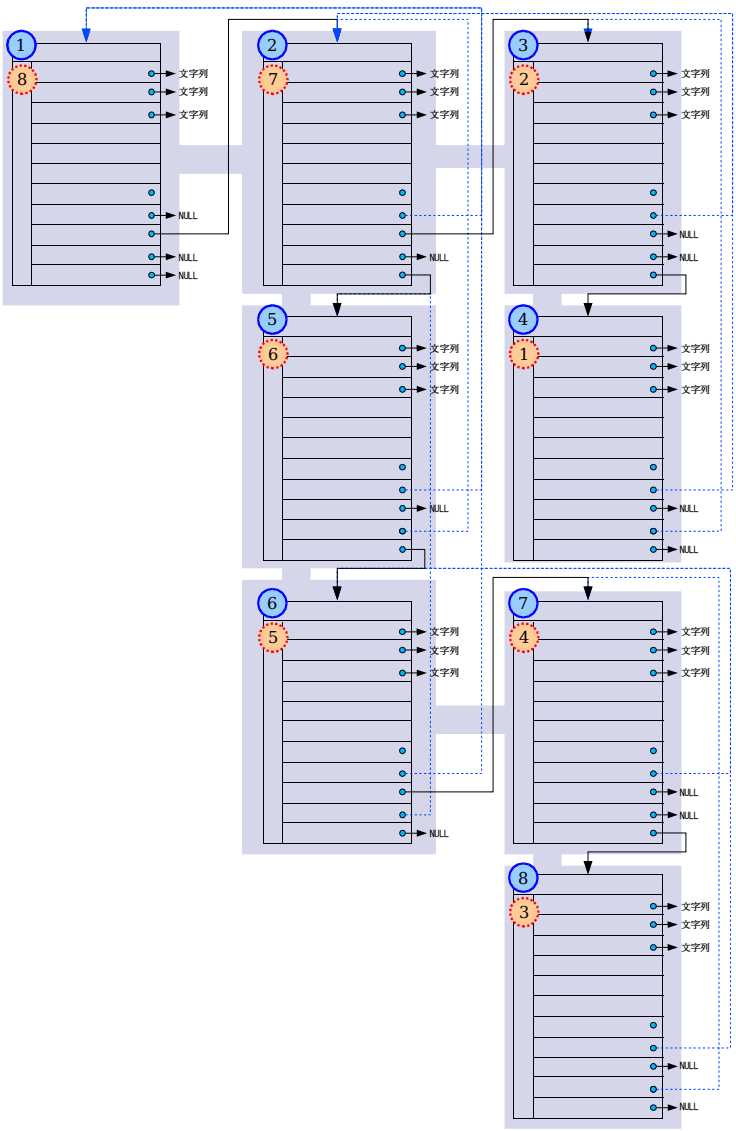
以下の表 7 は、表 6 を元にプログラム上で実際に管理する情報にまとめたものです。

表 7 widgetSet[] 配列の内容

	インクルード・ファイル	クラス名 クラス変数名	変数名/クラス変数/フォーマット文字列/引数情報/子部品の実現条件/部品種類 再構成可能判定/ツリー化判定/実現可能判定/部品作成
0	Composite.h	Composite compositeWidgetClass	&compositeWidgetClass/NULL/ NULL/NoChildMetaClass/NULL/NULL/NULL
1	Xaw/Box.h	Box boxWidgetClass	&boxWidgetClass/createNormal/createNormalArgs/MultiChild NormalWidget/hasFamily/partsExist/NULL
2	Constraint.h	Constraint constraintWidgetClass	&constraintWidgetClass/NULL/NULL/NoChild MetaClass/NULL/NULL/NULL
3	Xaw/Form.h	Form formWidgetClass	&formWidgetClass/createNormal/createNormalArgs/MultiChild NormalWidget/hasFamily/partsExist/NULL
4	Xaw/Dialog.h	Dialog dialogWidgetClass	&dialogWidgetClass/createNormal/createNormalArgs/MultiChild NormalWidget/hasParent/partsExist/NULL
5	Xaw/Viewport.h	Viewport viewportWidgetClass	&viewportWidgetClass/createNormal/createNormalArgs/MultiChild NormalWidget/hasFamily/partsExist/NULL
6	Xaw/Paned.h	Paned panedWidgetClass	&panedWidgetClass/createNormal/createNormalArgs/MultiChild NormalWidget/hasFamily/partsExist/NULL
7	Xaw/Tree.h	Tree treeWidgetClass	&treeWidgetClass/createNormal/createNormalArgs/MultiChild NormalWidget/hasParent/partsExist/NULL
8	Xaw/Porthole.h	Porthole portholeWidgetClass	&portholeWidgetClass/createNormal/createNormalArgs/OneChild NormalWidget/hasParent/partsExist/NULL
9	Shell.h	OverrideShell overrideShellWidgetClass	&overrideShellWidgetClass/createShell/createShellArgs/OneChild ShellWidget/hasFamily/partsExist/hasNormalChild
10	Xaw/SimpleMenu.h	SimpleMenu simpleMenuWidgetClass	&simpleMenuWidgetClass/createShell/createShellArgs/OneChild SimpleMenu/hasFamily/partsExist/NULL
11	Shell.h	TopLevelShell topLevelShellWidgetClass	&topLevelShellWidgetClass/createShell/createShellArgs/OneChild ShellWidget/hasFamily/partsExist/hasNormalChild
12	Intrinsic.h StringDefs.h	ApplicationShell applicationShellWidgetClass	&applicationShellWidgetClass/createApplication/createApplicationArgs/OneChild ShellWidget/hasChild/hasNoFamily/hasNormalChild
13	Shell.h	TransientShell transientShellWidgetClass	&transientShellWidgetClass/createShell/createShellArgs/OneChild ShellWidget/hasFamily/partsExist/hasNormalChild
14	Xaw/Simple.h	Simple simpleWidgetClass	&simpleWidgetClass/NULL/NULL/NoChild NormalWidget/NULL/NULL/NULL
15	Xaw/Grip.h	Grip gripWidgetClass	&gripWidgetClass/createNormal/createNormalArgs/NoChild NormalWidget/hasParent/partsExist/NULL
16	Xaw/Label.h	Label labelWidgetClass	&labelWidgetClass/createNormal/createNormalArgs/NoChild NormalWidget/hasParent/partsExist/NULL
17	Xaw/Command.h	Command commandWidgetClass	&commandWidgetClass/createNormal/createNormalArgs/NoChild NormalWidget/hasParent/partsExist/NULL
18	Xaw/MenuButton.h	MenuButton menuButtonWidgetClass	&menuButtonWidgetClass/createNormal/createNormalArgs/NoChild NormalWidget/hasParent/partsExist/NULL
19	Xaw/Repeater.h	Repeater repeaterWidgetClass	&repeaterWidgetClass/createNormal/createNormalArgs/NoChild NormalWidget/hasParent/partsExist/NULL
20	Xaw/Toggle.h	Toggle toggleWidgetClass	&toggleWidgetClass/createNormal/createNormalArgs/NoChild NormalWidget/hasParent/partsExist/NULL
21	Xaw/List.h	List listWidgetClass	&listWidgetClass/createNormal/createNormalArgs/NoChild NormalWidget/hasParent/partsExist/NULL
22	Xaw/Panner.h	Panner pannerWidgetClass	&pannerWidgetClass/createNormal/createNormalArgs/NoChild NormalWidget/hasParent/partsExist/NULL
23	Xaw/Scrollbar.h	Scrollbar scrollbarWidgetClass	&scrollbarWidgetClass/createNormal/createNormalArgs/NoChild NormalWidget/hasParent/partsExist/NULL
24	Xaw/StripChart.h	StripChart stripChartWidgetClass	&stripChartWidgetClass/createNormal/createNormalArgs/NoChild NormalWidget/hasParent/partsExist/NULL
25	Xaw/AsciiText.h Xaw/Text.h Xaw/TextSrc.h Xaw/TextSink.h Xaw/AsciiSrc.h Xaw/AsciiSink.h	Text asciiTextWidgetClass	&asciiTextWidgetClass/createNormal/createNormalArgs/NoChild NormalWidget/hasParent/partsExist/NULL
26	Xaw/Sme.h	Sme smeObjectClass	&smeObjectClass/createNormal/createNormalArgs/NoChild SmeObject/hasParent/partsExist/NULL
27	Xaw/SmeBSB.h	SmeBSB smeBSBObjectClass	&smeBSBObjectClass/createNormal/createNormalArgs/NoChild SmeObject/hasParent/partsExist/NULL
28	Xaw/SmeLine.h	SmeLine smeLineObjectClass	&smeLineObjectClass/createNormal/createNormalArgs/NoChild SmeObject/hasParent/partsExist/NULL

5.3 ツリー構造の分析

acconv には、アプリケーションのツリー構造を正しく編集するためにツリー構造自体を分析する必要がある。例えば、Box や Form, Paned など種類のウィジェットを削除すれば（ツリー構造上で）その子孫にあたる部品は、存在できないことになる。
このようなツリー構造に問題を起こさないように、構造的に問題になる部分を調べる必要があります。



左の図 16 は、ツリー構造を追いかけて処理するための処理で使用するポインタとその順番をまとめています。
今、現在使用している追跡方法は、PartsTopTrace() 関数、PartsBottomTrace() 関数で使用している方法です。（この関数は、24 ページに示しています。）

これらは、共に child_ptr, next_ptr の順で追跡していきます。しかし、関数コールの位置の関係で実際に処理される順番が変わります。つまり青点線のリンクは、追跡には使用していません。

図 *16* ツリー構造の追跡

このようなツリー構造を追跡して分析するために acconv では、リカーシブコールを用いています。実際にツリー構造分析をする関数を示す。

```
void PartsTopTrace(PartsTree *parts, void (*PartsProc)(PartsTree *parts));
void PartsBottomTrace(PartsTree *parts, void (*PartsProc)(PartsTree *parts));
int PartsSum(PartsTree *parts, int (*PartsProc)(PartsTree *parts));
XtPointer PartsSearch(PartsTree *parts, XtPointer (*PartsProc)(PartsTree *parts));
void PartsTopTrace2(PartsTree *parts, XtPointer client_data,
            void (*PartsProc)(PartsTree *parts, XtPointer client_data));
void PartsTopDownTrace(PartsTree *parts, Boolean (*PartsProc)(PartsTree *parts));
int PartsTraceSameParents(PartsTree *parts, XtPointer client_data,
            void (*PartsProc)(PartsTree *parts, XtPointer client_data));
```

ツリー構造分析をする関数

PartsTopTrace()では、次の追跡前に処理を行う。そのため見つけた順に処理を行います。

```
/*****************************************************************************
 *  PartsTopTrace：GUIパーツのツリー構造をトレースする
 *  形式
 *      void PartsTopTrace(PartsTree *parts,
 *                                  void (*PartsProc)(PartsTree *parts));
 *  引数
 *      PartsTree *parts;                       部品管理情報
 *      void (*PartsProc)(PartsTree *parts);
 *  説明
 *      指定処理を以下のように、上側からトレースしながら指定された関数によ
 *      り処理する。
《中略》
 *
 *****************************************************************************/
void PartsTopTrace(PartsTree *parts, void (*PartsProc)(PartsTree *parts))
{
   if(parts != NULL) {
      if(PartsProc != NULL) {
         PartsProc(parts);
      }
      if(parts->child_ptr != NULL) {
         PartsTopTrace(parts->child_ptr, PartsProc);
      }
      if(parts->next_ptr != NULL) {
         PartsTopTrace(parts->next_ptr, PartsProc);
      }
   }
}
```

ツリー構造を上位に位置する部品から追跡する *PartsTopTrace()* 関数

PartsBottomTrace()では、次の追跡後に処理を行う。そのため探索が終わった部品順に処理を行います。

```
/*****************************************************************************
 *  PartsBottomTrace: GUIパーツのツリー構造をトレースし指定処理を行う
 *  形式
 *      void PartsBottomTrace(PartsTree *parts,
 *                                  void (*PartsProc)(PartsTree *parts));
 *  引数
 *      PartsTree *parts;                       部品管理情報
 *      void (*PartsProc)(PartsTree *parts);
 *  返り値
 *
 *  説明
 *      指定処理を以下のように、下側からトレースしながら指定された関数によ
 *      り処理する。
《中略》
 *  バグ
 *        兄弟の部品と子供の部品の両方が同時に存在する場合は、アンリンクで
 *      きません。
 *        基本的には、子孫から順に開放してください。
 *
 *****************************************************************************/
void PartsBottomTrace(PartsTree *parts, void (*PartsProc)(PartsTree *parts))
{
   if(parts != NULL) {
      if(parts->child_ptr != NULL) {
         PartsBottomTrace(parts->child_ptr, PartsProc);
      }
      if(parts->next_ptr != NULL) {
         PartsBottomTrace(parts->next_ptr, PartsProc);
      }
      if(PartsProc != NULL) {
         PartsProc(parts);
      }
   }
}
```

ツリー構造を下位から部品から追跡する *PartsBottomTrace()* 関数

上記の様にソースコード上では、ごくわずかな違いが処理する順序を大きく変えています。処理の順番については、23 ページの図 16 に、PartsTopTrace() での処理順を①～⑧で、PartsBottomTrace() での処理順を①～⑧で示しています。

5.3.1 ツリーの枝葉の部分から根幹部分への順で調べる

先ほど例としてして示した、「Box や Form, Paned など種類のウィジェットの削除」ですが、acconv では、これらのウィジェット削除しようとすると、その子孫も含めて対象のウィジェットを破棄します。（しかも、削除の順番を考慮して）子孫を探索して下層の方から削除します。

上記の例である部品の削除では、以下のように PartsBottomTrace() 関数を用いている。

```
/*****************************************************************************
 *  partsDelete：部品を削除する
 *  形式
 *      void partsDelete(Widget w,
 *                          XtPointer client_data, XtPointer call_data);
 *  引数
 *      Widget w;               mguiDel/cmdMenuInfo.myWidget[2] ウィジェット
 *      XtPointer client_data;  現在選択中の部品情報
 *      XtPointer call_data;    未使用
 *  説明
 *
 *    ―――――――――――――――――――――――――――――――――――――――――
 *      部品の場所      ウィジェット名          備考
 *    ―――――――――――――――――――――――――――――――――――――――――
 *      ツール             mguiDel          直接のコールバック
 *    ―――――――――――――――――――――――――――――――――――――――――
 *      メニュー    cmdMenuInfo[2].myWidget partsDeleteInMenu から間接
 *    ―――――――――――――――――――――――――――――――――――――――――
 *
 *        指定処理を以下のように、下側からトレースしながら以下の順に部品を
 *      削除する。
 *
《中略》
 *
 *  参照事項
 *      partsDeleteOne() -- 指定された部品を削除する関数
 *
 *****************************************************************************/
void partsDelete(Widget w, XtPointer client_data, XtPointer call_data)
{
   PartsTree **parts = (PartsTree**)client_data;
   PartsTree *newParts;
   Widget parentTree;
   extern Widget mguiTree;

   if(parts != NULL) {
      if(*parts != NULL) {
         PartsBottomTrace((*parts)->child_ptr, partsDeleteOne);
         partsDeleteOne(*parts);
         *parts == NULL;
      }
   }
}
```

部品を削除する *partsDelete()* 関数

この関数では、指定された部品から子孫を削除するところを PartsBottomTrace() で行っている。

5.3.2 ツリーの根幹部分から枝葉の部分への順で調べる

ツリーの根幹部分からここの枝葉への順で処理していく例としては、以下のようなものがある。

- 部品ツリーを保存する
- テスト部品の生成確認のチェック
- ソースコード生成（保存ファイルと同じ順でウィジェット生成を行う）

5.3.2.1 部品ツリーの保存処理

ファイルメニュー→保存でファイル選択により、ファイルとして作成中のツリーを保存できますが、部品ツリーを保存（ファイル→保存でファイル選択）すると以下のようなファイルが作成される。

```
Xplanetconf  xplanetconf  [map]
       Paned  xplanetBase
              Box  xplanetMenu
                     MenuButton  fileMenu
              Box  xplanetTop
                     Toggle  altTarget
                     MenuButton  altTargetSelect
                     Label  altTargetLabel
              Form  xplanetForm
                     Toggle  curTarget
                     Label  curTargetLabel
                     MenuButton  curTargetIfType
                     MenuButton  curTargetIfSheme
                     MenuButton  curTargetAddressing
                     Label  curTargetAddressLabel
                     Label  curTargetNetmaskLabel
                     Label  curTargetGatewayLabel
                     Label  curTargetDomainLabel
                     Label  curTargetDNSServerLabel
                     Text  curTargetIPAddress
                     Text  curTargetNetmask
                     Text  curTargetGateway
                     Text  curTargetDomain
                     Text  curTargetDNSServer
```

acconv のツリー構造ファイル保存形式

これは、最上位の ApplicationShell から順を追って保存される。

以下は、ファイル保存の処理です。

```
/*****************************************************************************
 *  partsTreeSave: 部品ツリー情報を保存する
 *  形式
 *      void partsTreeSave(PartsTree *parts, FILE *fp)
 *  引数
 *      PartsTree *parts;       部品ツリーの根元
 *      FILE *fp;               保存するファイルのファイルポインタ
 *  説明
 *
 *****************************************************************************/
void partsTreeSave(PartsTree *parts, FILE *fp)
{
   save_fp = fp;
   while(parts->parent_ptr != NULL)
      parts = parts->parent_ptr;

   PartsTopTrace2(parts, (XtPointer)saveFileLine, treeList);
}
```

ファイル保存処理 *partsTreeSave()*関数

5.4 テストウィジェットの構築処理

この処理は、部品ツリーの追跡で「枝葉から根幹側へ」と「根幹側から枝葉へ」の 2 種類の追跡処理を行います。
acconv では、本物の Athena Widget を使ってテストウィジェットを作成します。しかし、ツリー構成のなかで「部品をおかしなところには位置したり」、「リソース設定が不十分だったり」、「他のウィジェットへ依存してたり」で部品の一部（場合によっては全部）がテスト不可なときもあります。なお作成可能な、テスト部品は表示されていなくても「生成」されています。

以下の関数は、可能なかぎり部品を使用してテストウィジェットを作成（再構築）するようにする処理です。

```
/*****************************************************************************
 *  partsRenewal: 部品の再構築を行う
 *  形式
 *      void partsRenewal(PartsTree *parts);
 *  引数
 *      PartsTree *parts;   部品管理情報
 *  返り値
 *
 *  説明
 *
 *****************************************************************************/
void partsRenewal(PartsTree *parts)
{
   int counter, length;
   Boolean autoMapFlag;

   while(parts->parent_ptr != NULL)
      parts = parts->parent_ptr;

   if(parts != NULL) {
      PartsBottomTrace(parts->child_ptr, partsDestroyWidget); /* テスト部品を破棄 */
      partsDestroyWidget(parts);

      partsCreateWidget(parts);
      PartsTopTrace(parts->child_ptr, partsCreateWidget);
      if(parts->partsWidget != NULL) {
         PartsTopDownTrace(parts, partsRealize);
         PartsBottomTrace(parts, partsMapping);
      }
   }
}
```

部品の再構築を行う *partsRenewal()*関数

まず、以前のテスト部品の破棄を下位側から破棄しています。破棄に使用している関数を示します。

```
/*****************************************************************************
 *  partsDestroyWidget: テスト実現ウィジェットの破棄
 *  形式
 *      void partsDestroyWidget(PartsTree *parts);
 *  引数
 *      PartsTree *parts;   部品管理情報
 *  返り値
 *
 *  説明
 *
 *****************************************************************************/
void partsDestroyWidget(PartsTree *parts)
{
   if(parts != NULL) {
      if(parts->partsWidget != NULL) {
         XtDestroyWidget(parts->partsWidget);
      }
      parts->partsWidget = NULL;
   }
}
```

テストウィジェットの破棄を行う *partsDestroyWidget()* 関数

このテストウィジェットの破棄を行う partsDestroyWidget() 関数を PartsBottomTrace() 関数から間接的に呼び出しと直接の呼び出しを行っています。

逆に、テストウィジェットの生成を行う partsCreateWidget() 関数は、直接この部品のテストをウィジェットを生成した後、PartsTopDownTrace() 関数で間接的に呼び出しています。

```
/**********************************************************************
 *  partsCreateWidget:
 *  形式
 *      void partsCreateWidget(PartsTree *parts);
 *  引数
 *      PartsTree *parts;   部品管理情報
 *  返り値
 *
 *  説明
 *
 **********************************************************************/
void partsCreateWidget(PartsTree *parts)
{
   if((parts != NULL)?(parts->athenaInfo != NULL):(False)) {
      if(parts->athenaInfo->remakable != NULL) {
         if(parts->athenaInfo->remakable(parts->athenaInfo, parts)) {
            if(parts->athenaInfo->widgetCreation != NULL) {
               parts->athenaInfo->widgetCreation(
                                    parts->athenaInfo, parts);
            }
         }
      }
   }
}
```

テストウィジェットの生成を行う *partsCreateWidget()* 関数

なお、この partsCreateWidget() 関数の中の parts->athenaInfo->remakable() 関数でウィジェットの再構成が可能か不可の判定を行っています。

acconv-0.1.2 → 0.1.3 へのバージョンアップでの変更点の「Form ウィジェットを作成しようとしたとき、エラーで終了する。（Form ウィジェットのテスト部品を作成するときの条件を変更）」ですが、この「Form ウィジェットのテスト部品を作成するときの条件を変更」とは、parts が Form のときの parts->athenaInfo->remakable の指す関数を変更したことになります。
つまり、widgetSet[] 配列の Form クラスの部分を変更したことになります。
Form クラスは、子ウィジェットがなかったらテストウィジェットの大きさが異常だと判定されて、結果として「異常終了」します。
このためテストウィジェットを作ることが acconv アプリケーションの異常終了につながってしまったと言うことになります。
このようにテストウィジェットに起因する異常終了は、現在の acconv 仕様上の欠陥と言えます。

5.5 ツリー構造からソースコードを作成する

ツリーデータからソースコードを作成する場合には、各ウィジェットを親から作成する事と部品データから、個々のウィジェットを作成する部分の 2 つの処理などが必要になります。

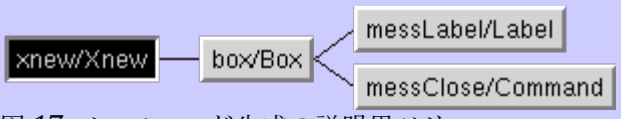


図 17 ソースコード生成の説明用ツリー

```
Xnew  xnew
   Box  box
      Label  messLabel
      Command  messClose
```

ソースコード生成を説明用ツリー構造

```
#include <X11/Intrinsic.h>
#include <X11/StringDefs.h>
#include <X11/Xaw/Box.h>
#include <X11/Xaw/Label.h>
#include <X11/Xaw/Command.h>

void main(int argc, char **argv)
{
      XtAppContext app_context;
      Widget xnew, box, messLabel, messClose;

      xnew = XtVaAppInitialize(
                        &app_context,
                        "Xnew",
                        NULL, 0,
                        &argc, argv,
                        NULL,
                        NULL);

      box = XtVaCreateManagedWidget(
                        "box",
                        boxWidgetClass,
                        xnew,
                        NULL);

      messLabel = XtVaCreateManagedWidget(
                        "messLabel",
                        labelWidgetClass,
                        box,
                        NULL);

      messClose = XtVaCreateManagedWidget(
                        "messClose",
                        commandWidgetClass,
                        box,
                        NULL);

      XtRealizeWidget(xnew);
      XtAppMainLoop(app_context);
}
```

インクルードの宣言

その他の埋め込まれる文字列
・ main() 関数の宣言
・アプリケーションコンテキストの宣言

ウィジェット変数の宣言

ウィジェットの生成

ウィジェットの実現

その他の埋め込まれる文字列
（メインループ, main() 関数終了の宣言）

acconv が生成するソースコード生成のサンプル

上記の様にソースコード生成には、以下のような要素が含まれます。

- インクルードの宣言の生成
- ウィジェット変数の宣言
- ウィジェットの生成
- ウィジェットの実現
- その他の埋め込まれる文字列（main()関数の宣言、アプリケーションコンテキストの宣言、メインループ XtAppMainLoop()関数）

```
/*****************************************************************************
 *  makeSource：C言語のソースファイルを作成する
 *  形式
 *      void makeSource(Widget w, XtPointer client_data, XtPointer call_data);
 *  引数
 *      Widget w;                   未使用
 *      XtPointer client_data;      部品情報
 *      XtPointer call_data;        未使用
 *  説明
 *
 *****************************************************************************/
void makeSource(Widget w, XtPointer client_data, XtPointer call_data)
{
   int counter;         /* ループカウンタ */
   int numOfClass;      /* クラス情報を持つウィジェット数を格納する変数 */
   int length;          /* 文字列長 */
   int *classArray;     /* クラス別で使用の有無を調べるための配列 */
   int argumentsInfo[] = {ArgWidget, NoArg};   /* XtRealize() の引数情報 */
   char stringBuffer[80];  /* 文字列バッファ */
   PartsTree *parts =      /* 部品情報 */
         *(PartsTree **)((struct menu_data*)client_data)->client_data;
   extern Widget mguiSourceShell, mguiSourceArea, mguiSourceTools;
   extern Widget mguiPopdown, mguiSave, mguiLoad, mguiSourceFile, mguiSourceText;

   XtPopup(mguiSourceShell, XtGrabNonexclusive);
   while(parts->parent_ptr != NULL)
      parts = parts->parent_ptr;

   /***********************************************
    *  パブリック・インクルードファイル部分の作成
    ***********************************************/
   numOfClass = sizeof(widgetSet)/sizeof(widgetSet[0]);
   classArray = (int*)XtMalloc(sizeof(int) * numOfClass);
   if(classArray != NULL) {
      /*
       *
       */
      for(counter = 0;counter < numOfClass;++counter)
         *(classArray+counter) = False;
      PartsTopTrace2(parts, (XtPointer)classArray, partsCollection);
      PartsTopTrace2(parts, (XtPointer)classArray, partsInclude);
      XtFree((char*)classArray);
      sourceList = stringConnect(sourceList,
                stringsAlloc("\nvoid main(int argc, char **argv)\n{"));
      sourceList = stringConnect(sourceList,
                     stringsAlloc("\tXtAppContext app_context;"));
   }
   /**********************
    *  Widget 変数の定義
    **********************/
   stringBuffer[0] = '\0';
   PartsTopTrace2(parts, (XtPointer)stringBuffer, partsVariable);
   length = strlen(stringBuffer);
   if(length > 0) {
      sprintf(stringBuffer, "%.*s;\n", length-1, stringBuffer);
      sourceList = stringConnect(sourceList, stringsAlloc(stringBuffer));
   }
   /******************
    *  Widget の構築
    ******************/
   PartsTopTrace(parts, partsCreateCoding);
   /**********************
    *  Widget の実態作成
    **********************/
   partsCoding(parts, "\tXtRealizeWidget(%s);", argumentsInfo);
   /*printf("\tXtAppMainLoop(app_context);\n}");*/
   sourceList = stringConnect(sourceList,
                stringsAlloc("\tXtAppMainLoop(app_context);\n}\n"));
   XtVaSetValues(mguiSourceText,
                XtNstring, stringsLink2string(sourceList), NULL);
   stringRelease(sourceList);
   sourceList = NULL;
}
```

インクルードの宣言

その他の埋め込まれる文字列
・ main() 関数の宣言
・アプリケーション
　コンテキストの宣言

ウィジェット変数の宣言

ウィジェットの生成

ウィジェットの実現

その他の埋め込まれる文字列
・メインループ
・ main() 関数終了の宣言

C 言語のソースファイルを作成する *makeSource()*関数

5.5.1 部品管理情報からインクルードファイルの部分を作成する

使用している部品から必要なヘッダーファイル[*1]を調べインクルードする行を生成する部分。プリプロセッサ[*2]で処理される部分です。

```
    int numOfClass;     /* クラス情報を持つウィジェット数を格納する変数 */
    int *classArray;    /* クラス別で使用の有無を調べるための配列 */

    /***********************************************
     *  パブリック・インクルードファイル部分の作成
     ***********************************************/
    numOfClass = sizeof(widgetSet)/sizeof(widgetSet[0]);
    classArray = (int*)XtMalloc(sizeof(int) * numOfClass);
    if(classArray != NULL) {
       /*
        *
        */
       for(counter = 0;counter < numOfClass;++counter)
          *(classArray+counter) = False;
       PartsTopTrace2(parts, (XtPointer)classArray, partsCollection);
       PartsTopTrace2(parts, (XtPointer)classArray, partsInclude);
       XtFree((char*)classArray);
    }
```

*makeSource()*関数内部のインクルードファイル作成部分

classArray は、int 型の部品の種類数分の配列としてメモリを確保、初期化して使用します。そして、２つの PartsTopTrace2() で使用している部品の種類を調べます。
最初の PartsTopTrace2(parts, (XtPointer)classArray, partsCollection)では、使用している部品の種類を classArray 配列にまとめています。次の PartsTopTrace2(parts, (XtPointer) classArray, partsInclude)では、再度、上位から（といってもツリー構造ファイル保存）順で最初にあらわあれた部品種に必要なインクルードファイルからソースにコードのバッファに書き込まれます。
なおここで必要なヘッダーファイルと判定されるファイルは、基本的に「パブリックインクルードファイル」と呼ばれるものです。オブジェクト指向言語では、メンバー変数は、外部へ公開されているものとオブジェクトの内部だけで使用できるものがあります。
パブリックインクルードファイルと呼ばれるヘッダーファイルでは、公開されているメンバー変数を取得したり設定したりできるようになっています。
これとは逆にクラス内部のソースの場合は、プライベートメンバーを使用するためのプライベートインクルードファイルを使用するようにします。

X Toolkit では、この使い方を無視してプライベートインクルードを外部から使用することも言語仕様上規制が無いため不可能ではないです。しかし、本来操作できないはずの変数への変更されて異常な動作になったり、管理があいまいになるのことからでバグを埋め込む可能性も大きくなります。このためプライベートインクルードファイルは使用すべきではありません。

*1 C 言語のプリプロセッサによってコンパイル前にソースコードに挿入して取り込まれるファイル。インクルードファイルと同義。
*2 プリプロセッサとは、コンパイルの前にソースコードに対して処理を行う部分。共通に使われるプロトタイプ宣言やマクロの定義などが行われている。

5.5.2 Widget 変数宣言部分を作成する

部品ツリー構造から、全ての変数名を取得して列記することで、Widget 変数宣言を作成します。

```
    /**********************
     *  Widget 変数の定義
     **********************/
    stringBuffer[0] = '\0';
    PartsTopTrace2(parts, (XtPointer)stringBuffer, partsVariable);
    length = strlen(stringBuffer);
    if(length > 0) {
        sprintf(stringBuffer, "%.*s;\n", length-1, stringBuffer);
        sourceList = stringConnect(sourceList, stringsAlloc(stringBuffer));
    }
```

makeSource()関数内部の Widget 変数宣言作成部分

acconv で作成する Widget 変数は、main()関数のローカル変数として作成されます。

```
/*****************************************************************************
 *  partsVariable：ウィジェット生成に必要な変数を宣言する
 *  形式
 *      void partsVariable(PartsTree *parts, XtPointer client_data);
 *  引数
 *      PartsTree *parts;       作成する部品の情報
 *      XtPointer client_data;  変数宣言の行の文字列
 *  説明
 *
 *
 *****************************************************************************/
void partsVariable(PartsTree *parts, XtPointer client_data)
{
    int length;
    char *variable_ptr;
    char *variableDefinition = (char*)client_data;

    length = strlen(variableDefinition);
    variable_ptr = variableDefinition + length;
    if((strlen(variableDefinition) + length) > 60) {
        sprintf(variable_ptr, "%s %s;",
                ((length == 0)?("\tWidget"):("")), parts->widgetName);
        /*printf("%s\n", variableDefinition);*/
        sourceList =
                stringConnect(sourceList, stringsAlloc(variableDefinition));
        *variableDefinition = '\0';
    } else {
        sprintf(variable_ptr, "%s %s,",
                ((length == 0)?("\tWidget"):("")), parts->widgetName);
    }
}
```

Widget 変数宣言を作成する partsVariable()関数

変数宣言の中の変数部分の文字列長が 60 バイトを越えれば、新しい行を作成します。越えなければ、同じ行に追加します。

5.5.3 部品管理情報からウィジェット生成のコードの部分を作成する

個々の部品を作るには、makeSource()関数から「部品ツリー全体を追跡する PartsTopTrace()関数」に「個々の部品の生成コード作成する partsCreateCoding()関数」を呼び出すようにさせます。

```
   /******************
    *  Widget の構築
    ******************/
   PartsTopTrace(parts, partsCreateCoding);
```

*makeSource()*関数内部のウィジェット生成のコードを作成する部分

```
/*****************************************************************************
 *  partsCreateCoding: Widget 作成部分をコーディングする
 *  形式
 *      void partsCreateCoding(PartsTree *parts);
 *  引数
 *      PartsTree *parts;   作成する Widget に関する情報
 *  説明
 *
 *****************************************************************************/
void partsCreateCoding(PartsTree *parts)
{
   if(parts != NULL && parts->athenaInfo != NULL &&
                       parts->athenaInfo->argTypes != NULL) {
      partsCoding(parts,
            parts->athenaInfo->instance, parts->athenaInfo->argTypes);
   }
}
```

一つの部品に対応するウィジェット生成のコードを生成する *partsCreateCoding()*関数

```
/*****************************************************************************
 *  partsCoding: Widget に関してフォーマット化文字列と引数からコーディング
 *  形式
 *      void partsCoding(PartsTree *parts,
 *                               const char *format, int *argumentsInfo);
 *  引数
 *      PartsTree *parts;   作成する Widget に関する情報
 *      const char *format; フォーマット化文字列（sprintf() のもの）
 *      int *argumentsInfo; 引数情報を記した整数配列
 *  説明
 *      Widget に関してフォーマット化文字列と引数情報からコーディングする
 *
 *****************************************************************************/
void partsCoding(PartsTree *parts, const char *format, int *argumentsInfo)
{
   int counter;
   int argumentType;
   char *arguments[10];
   char stringBuffer[1024];

   /*******************************
    *  関数の引数部分を初期化する
    *******************************/
   for(counter = 0;counter < 10;++counter)
      arguments[counter] = "";

   /*************************
    *  関数の引数部分を作成
    *************************/
   if(parts != NULL && format != NULL && argumentsInfo != NULL) {
      for(counter = 0;*(argumentsInfo+counter) != NoArg;++counter) {
         argumentType = *(argumentsInfo+counter);
         /*printf("argument#%d's type = %d.\n", counter, argumentType);*/
         if(argumentType > NoArg && argumentType < VariableTypeNum) {
            arguments[counter] = argumentArrayProc[argumentType](parts);
         }
      }
      /***************************************
       *  Widget の作成部分をコーディングする
       ***************************************/
      sprintf(stringBuffer, format, arguments[0], arguments[1],
            arguments[2], arguments[3], arguments[4], arguments[5],
            arguments[6], arguments[7], arguments[8], arguments[9]);
      sourceList = stringConnect(sourceList, stringsAlloc(stringBuffer));
   }
}
```

部品情報に関してフォーマット化文字列と引数情報からコード作成をする *partsCoding()*関数

前のページ（33 ページ）の partsCoding()関数についてですが、この関数は、以下の形式です。
void partsCoding(PartsTree *parts, const char *format, int *argumentsInfo)

- PartsTree *parts.....部品情報
- const char *format.....printf() 系関数のフォーマット化文字列
- int *argumentsInfo.....int 型配列を示す関数。

この partsCoding()関数は、以下のようなデータを処理します。

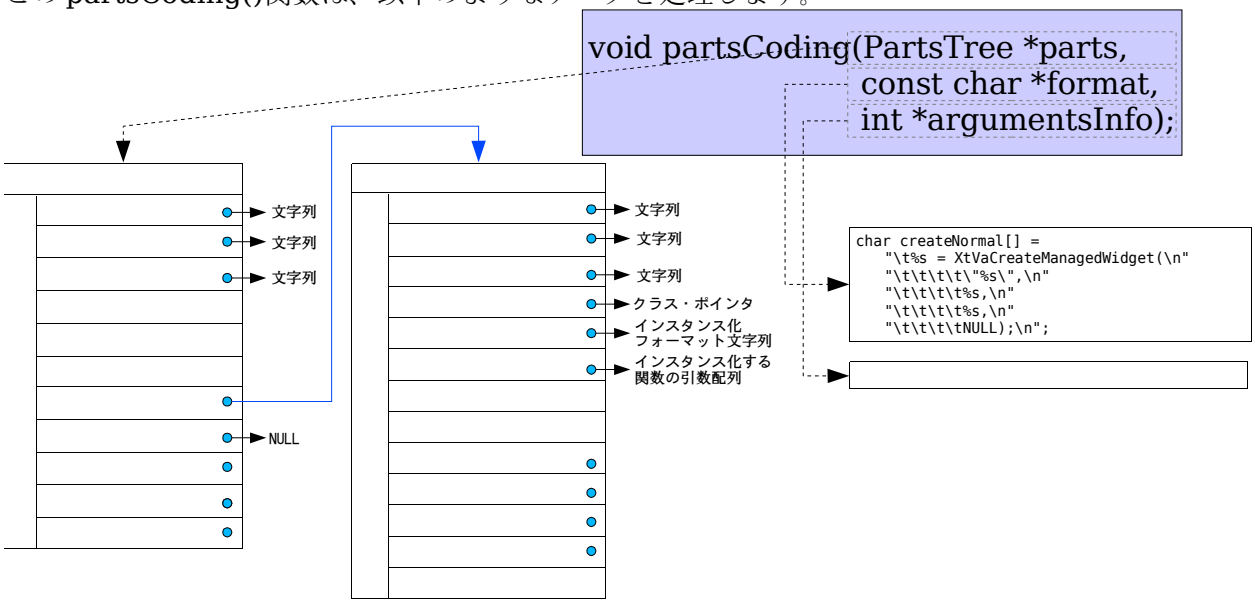


図 *18 partsCoding()*関数

partsCoding()関数の中では、format 引数と argumentsInfo 引数からコードを生成します。

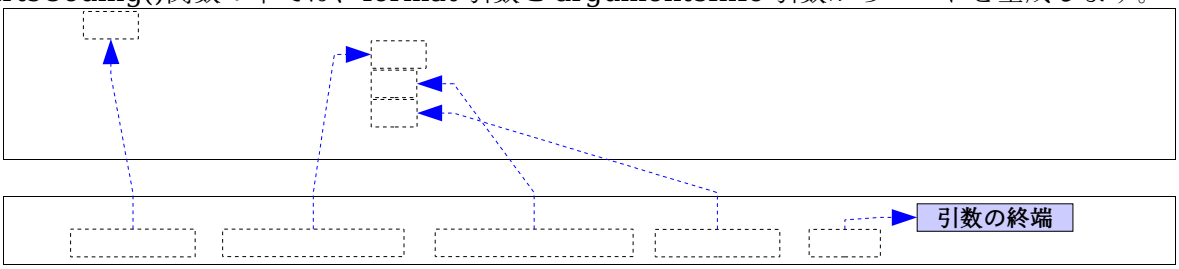


図 *19 partsCoding()*関数のフォーマット化文字列と引数情報の対応

argumentsInfo の引数は、中の配列の各要素は、argumentArrayProc[]配列に対応しています。
argumentArrayProc[]配列は、部品管理情報から引数の文字列を得る関数へのポインタ配列です。
実際には、配列の添え字によって獲得したい引数の種別がかわります。
配列の添え字は、NoArg(0) 以上で、VariableTypeNum 未満になります。

表 8 argumentArrayProc[]配列の内容

配列の添え字	実際に呼ばれる関数	意味
NoArg	char *noArg(PartsTree *parts);	引数なし
ArgWidget	char *argWidget(PartsTree *parts);	ウィジェット変数名
ArgWidgetName	char *argWidgetName(PartsTree *parts);	ウィジェット名
ArgWidgetClass	char *argWidgetClass(PartsTree *parts);	クラス変数名
ArgParent	char *argParent(PartsTree *parts);	親ウィジェット変数名
ArgAppContext	char *argAppContext(PartsTree *parts);	アプリケーションコンテクスト変数名
ArgApplicationClass	char *argApplicationClass(PartsTree *parts);	アプリケーションクラス名
VariableTypeNum	---	添え字範囲オーバー値（範囲外検出用）

5.5.4 ウィジェットの実現処理部分のコード作成する

この処理は、最上位のウィジェットに対して XtRealizeWidget()を呼び出すだけです。

```
/**********************
 *  Widget の実態作成
 **********************/
partsCoding(parts, "\tXtRealizeWidget(%s);", argumentsInfo);
```

*makeSource()*関数内部の *Widget* を実現処理を作成する部分のコード

ここで、arumentsInfo は、関数冒頭で以下のように定義されています。

```
int argumentsInfo[] = {ArgWidget, NoArg};   /* XtRealize() の引数情報 */
```

つまり、XtRealiseWidget()に単一の最上位のウィジェットの「ウィジェットの変数」を渡すようになっています。

partsCoding() 関数については、33 ページのソースコードや 34 ページの図 19 の説明を参照してください。

6 バグ

acconv では、アプリケーション用に作成されたリソースファイルを部品構成ファイルと共に読み込もうとします。しかし、読み込まれたリソースは、acconv 自体にも影響をあたえてしまいます。 場合によっては、致命的な影響をおよぼします。 また、国際化への対応が不十分です。 現行の acconv は、非常に不安定です。これへの対策は、こまめに保存することしかありません。

7 今度の課題

山積みになってます。

8 Special Thanks

個人名をあげて書くことはできないのですが、以下の方々に感謝します。
- C 言語, X Window System, X Toolkit, Athena Widget の開発などをされた方々。
- Xaw3D などの Athena Widget の代替ライブラリを製作された方々。
- 全ての Linux の開発にかかわった方々（私としては、特に Linux/TOWNS, X TOWNS 関係者）

9 ライセンス

Nobuyuki Maruichi
E-mail: HFD03621@nifty.ne.jp

Appendix A 関数一覧

以下の関数一覧は、以前の acconv.tex から再構成したため古い仕様のものが混ざっているかも知れません。

表 9 acconv 内部関数一覧

関数名	説明
PopupNamingDialog	ディスク・データ登録用のダイアログのポップアップ
NamingDialogSet	ディスク・データの登録
NamingDialogPopdown	ウィジェット名の命名用のウィンドウをポップダウン
PopdownAncestorShell	ウィジェットの最も近い先祖のシェルをポップダウン
hasParent	親ウィジェットの存在を調べる
hasChild	子ウィジェットの存在を調べる
hasFamily	親／子ウィジェットの存在を調べる
hasNoFamily	親／子ウィジェットの存在を調べる
hasNormalChild	普通の子ウィジェットの存在を調べる
partsExist	選択されているパーツの存在を調べる
alwaysMakable	常にウィジェット作成が可能なことを返す関数
alwaysDismakable	常にウィジェット作成が不可能なことを返す関数
widgetCreation	普通のウィジェットを作る
shellCreater	Shell ウィジェットを作成する
partsMapping	ウィジェットのマップ／アンマップを設定する
makableObject	作成可能なウィジェットのリストを作成する
partsPopup	指定されたシェル部品をポップアップする
partsPopdown	指定されたシェル部品をポップダウンする
partsTreeSave	部品ツリー情報を保存する
partsTreeCopy	部品ツリー情報をコピーバッファに保存する
stringAlloc	１行分のメモリを確保する
stringFree	文字列用のメモリを解放する
stringRelease	文字列用のメモリを解放する
stringsAlloc	１行以上のメモリを確保する
stringConnect	文字列リンクのメモリを解放する
partsCoding	Widget に関してフォーマット化文字列と引数からコーディング
partsCreateCoding	Widget 作成部分をコーディングする
stringsLinkLength	文字列リンクストリストの文字数をカウントする
stringsLink2string	文字列リンクを文字列に変換
partsLabelSet	ツリー部品に指定された形式でラベルを張り付ける
partsWidgetVariable	
partsWidgetName	
partsClassNames	
partsWidgetAndClassNames	
partsResourceBox	
samenameCheck	リネームシェルを作成します
variableNaming	ウィジェット変数名の名付けを行う
setRenames	
setNames	
createRenameShell	リネームシェルを作成します

図の索引

表の索引

ソースコード索引

索引