Python Mega Widget(Pmw) ライブラリ*

gregm@iname.com

January 23, 2001

Abstract

Pmw[1]は Python/Tkinter を使って GUI を作成するための高度な複合 widget (megawidget) の集合体である。Pmw はまたそのような複合 widget の作成を容易にするための枠組みをも提供する。Pmw をつかうことで、柔軟でカスタマイズの容易なアプリケーションを簡単につくれるようになる。

1 主な機能

Python/Tkinter は GUI(Graphical User Interface) を作成するうえで欠かせない基本的な部品 (widget) を提供する。実用的な GUI を作成するためには、これらの基本的な widget を組み合わせた複合 widget を作成することが必要となる。これらの複合 widget を使うことで、GUI 全体の統一性を保つことが可能となる。Pmw(Python Mega Widget) はこのような目的のために作成された複合 widget の集合である。また Pmw はさらに高度な Widget を作成するための枠組みとして使うことも出来る。Pmw は簡単に使えまた、柔軟に設定を変更できる。Pmw は以下の部分から構成される。

- megawidget を作成するための枠組みとなる幾つかの基本クラス
- 基本クラスの上に構築された柔軟で拡張可能な widget のライブラリ。ライブラリにはボタンボックス、ComboBox, メッセージダイアログ等が含まれている。
- Pmw が最初に import されたときにロードされる lazy importer/動的 loader. この lazy importer/動的 loader によって Pmw に含まれるすべてのクラスに Pmw. 前置子を使ってアクセス出来るようになる。また必要な副モジュールだけをロードすることでモジュールのロードの時間を短縮する。
- すべての megawidget とそのオプション、メソッド、部品を説明した参考文書。基本クラスおよび幾つかの megawidget については、完全な説明書が付属する。
- Tkinter と Pmw のすべての widget をテストするための枠組みとテストアプリケーション

*この文章は、Pmw0.7 とともに配付される html 形式の DOC ファイルの抄訳である。日本語訳: 高エネルギー加速器研究機構, 加速器第二研究系山本昇

[1]http://www.python/org

- megawidget の機能を紹介するデモンストレーション
- Blt, Tk にグラフなどの機能を付け加える拡張、へのインタフェイス

Pmw に含まれる megawidget は Tkinter の基本的な widget と同じようなやり方で使うことが出来る。megawidget と基本的な widget が混在するアプリケーションを混乱無く作成することが可能である。Pmw の megawidget はまた、基本的な widge と同じく、継承や複合によって拡張が可能である。

Pmw の megawidget とその枠組みは Tkinter ではよく使われる widget の組み合わせを高次の抽象化された層で置き換える。これによって開発するプログラムは簡単になり、読みやすくまた保守を容易にする。またプログラムの後での変更も容易になる。Pmw-megawidget の拡張性を利用は、以前のコードを megawidget 化することで開発者のコードの再利用を促進する。

2 始めてみよう

この章では Pmw の入手法、インストールの方法、そしてデモンストレーションとテストの実行方法を説明する。

2.1 準備

Pmw.0.7 は _tkinter と Tkinter が組み込まれた Python1.5 が必要である。 組み合わせる Tck/Tk ライブラリのバージョンは 4.2 または 8.0 が必要である。 Tk の BLT 拡張が組み込まれていれば、Pmw は BLT の busy コマンドを modal dialog を表示するさいにカーソルを時計カーソルとするために使う。また Pmw.Blt を使うことで、BLT の Graph, Vector, busy の各コマンドが Python から利用可能となる。

2.2 Pmw の入手とインストール

Pmw の配付キットは ftp.dscpl.com.au から ftp で入手可能である。

```
URL:ftp://ftp.dscpl.com.au/pub/pmw/Pmw.0.7.tar
```

最新版は Pmw.0.7 である（1998 年 6 月 2 日配布）。

配付ファイルは、Unix では次のコマンド

```
gunzip Pmw.0.7.tar.gz
tar xvf Pmw.0.7.tar
```

で展開出来る。Macintosh や Windows でも同様のユーティリティを使用して展開する。

これらのコマンドによって、Pmw と云う名前のディレクトリ（あるいは Folder）の下に Pmw のファイルが展開される。このディレクトリあるいはこのディレクトリへのシンボリック・リンクを site-packege ディレクトリに入れておくことで、Pmw が Python から利用可能になる。

もし site-pacakges へのかきこみ許可をあなたが持っていないのなら、Pmw ディレクトリはあなたの環境での PYTHONPATH に含まれているディレクトリのどこに置くことも

可能である。どうしてもこれらのところにPmwを置くことが出来なければPmwを展開した親のディレクトリをPYTHONPATHあるいはsys.pathに付け加えることで、Pmwが使用可能となる。

2.3 ドキュメント

Pmwディレクトリ下のdocディレクトリはそのバージョンに関するすべてのDocumentを含んでいる。index.htmlにはすべてのドキュメントのリストが含まれている。このdocディレクトリの文書はまた、Pmwの公式ホームページ (URL: http://www-acc.kek.jp/WWW-ACC-exp/KEKB/control/Activity/Python/Pmw-doc/starting.html) から入手可能である。

2.4 デモンストレーションとテスト

Pmwでなにが可能であるかの概略を知るためにはデモンストレーションとテストを実行し, デモンストレーションプログラムのソースコードをみてみるのがよい方法である。Pmwで可能多くの機能をみてみるには,demosディレクトリのAll.pyを実行してみよ。

デモンストレーションやテストを実行するためにPmwをインストールする必要はない。単にtestsディレクトリやdemosディレクトリにcdして, All.pyを実行するだけでよい。

2.5 バグ

Tk8.0でのみ現れる小さなバグが知られている。ダイアログボックスがアクティブで、かつダイアログボックスがカーソルの上にポップアップし、ダイアログボックスがデフォルトボタンを持つ場合、Returnキーを押してもデフォルトボタンの動作が実行されない。マウスを動かして、カーソルをいったんダイアログの外に出してから再びダイアログにもどすとReturnキーは通常どおり動作する。　　colorは米語式に綴られているが、initialiseは英語式である。　　あと一つか二つのバグがあることは確かである。あなたが教えて下さると確信している。

3 Pmw メガウイジェットの使い方

この章では、Pmwメガウイジェットの特徴とその使用法を簡単に説明する。例を使ってPmw メガウイジェットに共通な特徴を説明する。夫々のPmwメガウイジェットに特有な機能についてはリファレンス・マニュアルを見て欲しい。Pmw メガウイジェット一般に共通な機能の完全な記述はPmw.MegaArchetypeのリファレンス・マニュアルにある。Pmw.MegaArchetypeはすべてのの Pmw メガウイジェットの共通の祖先となるクラスである。 Pmwのdemoディレクトリには、さまざまな例が収められている。

最初に示すメガウイジェットの例はカウンタである。このウイジェットは一つのエントリ・フィールドと二つの小さなアロー・ボタンを持っている。ユーザは数値を直接このエントリ・フィールドに入れることも出来るし、アロー・ボタンを使って値を増減することもできる。これとその他のメガウイジェットをあなたの道具箱に入れておくことで、あなたのアプリケーションのグラフィカル・ユーザインタフェイスを最適なものにできます。

3.1 Pmw の初期化

この節の例題を実行するために、Pmw の lib ディレクトリが sys.path に含まれている事を確認して下さい。もし sys.path に Pmw の lib ディレクトリが含まれていない場合には、このディレクトリを追加して下さい。

次の二行はすべての例題の実行に先立って Python の中で実行される必要があります。これらの文は Pmw モジュールを import し、初期化します。Pmw.initilaise() についての詳しい説明は Pmw 関数リファレンス・マニュアルをご覧下さい。

```
import Pmw
root = Pmw.initialise()
```

Tkinter と Pmw の初期化を制御するために、次の形を使うことも可能である。

```
import Tkinter
root = Tkinter.Tk()
import Pmw
Pmw.initialise(root)
```

3.2 メガウイジェットの生成

さて初期化の儀式も終わったので、カウンタ・メガウイジェット (Pmw.Counter) を作り出す準備が整いました。デフォルトの設定を持つカウンタ・メガウイジェットは次の Python の文で作り出せる。

```
counter1 = Pmw.Counter()
counter1.pack(padx = 10, pady = 10)
counter1.mainloo() # make it running and visible
```

さあ、現れたカウンタ・メガウイジェットのエントリ・フィールドに数字をいれて、アロー・ボタンをクリックして数字が増減するのをみて下さい。結果はこのようなものになります。

この例ではカウンタ・ウイジェットは root ウィンドウの子供として作成されました。もし、カウンタ・ウイジェットを他のウィンドウの子供として作りたいなら、(例えば Tkinter の Frame の中に埋め込まれたウイジェットを作成する) 親となるウイジェットを生成子の引数として与えます。

```
counter1a = Pmw.Counter(frame)
```

3.3 メソッド

いったんメガウイジェットが作られたなら、そのメガウイジェットのもつ method を、Tk-widget と同じように、呼び出すことが可能です。次の文はカウンタウイジェットに値を設定しそれを一増やします。

```
counter1.setentry(41)
counter1.increment()
```

3.4 オプション

その他の（Tkinterの）ウイジェットと同じように、メガウイジェットは使用目的に応じて変更可能なオプションを持っている。メガウイジェットのオプションを使うことで、ユーザはその見え方や振る舞いを変更できる。カウンタ・メガウイジェットもそのようなオプションを幾つかもっている。*datatype* はカウンタがどのように数を増減するかを指定する。例えば、実数として、あるいは整数、時間、日付などとして増減させることができる。*datatype* のデフォルト値は*'numeric'* である。*'numeric'* の datatye ではカウンタは整数値が入力されるものと期待する。また増減はカウンタの数値全体を対象とする。

increment はまた別のオプションである。このオプションはカウンタの増減ボタンが押された時に、どれだけの単位をカウンタの数値に足すあるいは引き去るかを指定する。これらのオプションを使うことで、時間を数えるカウンタを作ることができる。表示は"00:00:00"の形式で、増減の単位は分とする。このカウンタをつくるための Python プログラムは次の様になる。

```
counter2 = Pmw.Counter(
     datatype = 'time',
     increment = 60)
counter2.setentry('00:00:00')
counter2.pack(padx = 10, pady = 10)
```

（初期値を設定するための setentry() メソッドを使っていることに注意。）

多くのメガウイジェットオプションは configure() メソッドを使って変更可能である。例えば、上で作成したカウンタ・ウイジェットの *increment* を 10 分とするのには、

```
counter2.configure(increment = 60 * 10)
```

とすればよい。

3.4.1 初期化のオプション

幾つかのメガウイジェットオプションは初期化時にのみ設定できる。これらのオプションは configure() メソッドを使って変更できない。しかし其の設定値を後で読み出すことは可能である。例えば、カウンタには初期化オプション *orient* がある。このオプションで増減ボタンが左右にある（*'horizonta'*）か上下にあるか（*'vertical'*）を指定できる。上下に増減ボタンがある数値カウンタは、

```
counter3 = Pmw.Counter(orient = 'vertical')
counter3.pack(padx = 10, pady = 10)
```

でつくり出すことができる。

3.4.2 オプションの値を読み出す

メガウイジェットのオプションの現在値は、通常の Tkinter　ウイジェットと同じように、読み出すことが可能である。次の例はカウンタオプションの幾つかをプリントするものである。

```
print counter3.cget('increment')
     --> 1
print counter3.configure('orient')
     --> ('orient', 'orient', 'Orient', 'horizontal', 'vertical')
```

Tk ウイジェットのオプションの現在値を読み出すと、値は常に string 型として返される。Pmw メガウイジェットでは、これとは異なり、オプションが設定された時のオブジェクトへの参照が値として返される。言い換えれば、cget() の値として返される値の型は常に String とは限らない。上記の例では cget('increment') の返した値は（文字データではなく）整数の値である。

3.5 部品 (Componets)

メガウイジェットは部品（componets）と呼ばれるウイジェットから構成されている。夫々の部品は論理的な名前で区別され、単純な Tkinter のウィジェットあるいはそれ自身がメガウイジェットである。Pmw はユーザにメガウイジェット自身のメソッドとオプションによる制御だけでなく、各部品のメソッドとオプションを操作することを許している。部品を直接に操作するために、componet() メソッドを使う。例えば、メガウイジェット *mega* の部品 *comp* のメソッド *doit* を呼ぶには、

```
mega.component('comp').doit()
```

とすればよい。部品のオプションを取り扱うための省略記法として *componet_option* を用いることが許されている。これを使うと例えば増減ボタンの夫々がことなったバックグラウンド色をもつ様に設定可能である。（それらの部品は、*downarrow* および *uparrow* と呼ばれる。）

```
counter2.configure(
    downarrow_background = 'green',
    uparrow_background = 'red')
```

3.5.1 hull

すべてのメガウイジェットは Pmw の基本クラスによって作られる containing widget の中にしまわれている。通常この containing widget は Tkinter の Frame ウイジェットである。トップレベルのウィンドウになるメガウイジェットの containing widget は Tkinter の Toplevel ウイジェットとなっている。containing widget は部品 *hull* として引用される。

3.5.2 内部部品 ()

メガウイジェットの中には Dialog や LabeledWidget 等のように、ユーザがその他のウイジェットを埋め込むことができる Frame を部品として持つものがある。この Frame は部品ではあるが、interior() メソッドを通じて取り扱うこともできる。Pmw.MegaToplevel ウイジェットおよび Pmw.MegaWidget については interior ウイジェットは hull ウイジェットと同じウイジェットになる。その他のメガウイジェットでは hull は外側のウイジェットを包むウイジェットで、interior は中身が空の Frame となる。この Frame(interior) に新たなウイジェットを組み入れることで、メガウイジェットを拡張することができる。

3.5.3 副部品 (sub components) と別名 (aliases)

メガウイジェットの部品はそれ自身がメガウイジェットであってよくそれ自身が自分の（副）部品を持っていてもよい。これらの副部品は *component_sub-componet* の形式で引用出来る。例としてカウンタウイジェットの *entryfield* をあげてみる。*entryfield* はそれ自身が Pmw.EntryField メガウイジェットである。入力された値のチェックば Pmw.EntryField メガウイジェットが行っている。Pmw.EntryField メガウイジェットは部品 *entry* として、Tkinter.Entry ウイジェットをもってる。したがって、カウンタ・メガウイジェットに含まれる Tkinter.Entry ウイジェットのバックグランドの色は、

```
counter2.configure(entryfield_entry_background = 'yellow')
```

多くの部品の正式な名前 (*entryfield_entry* など) は短い別名を持っている。上の例と等価な文として、

```
counter2.configure(entry_background = 'yellow')
```

を使うことも出来る。

3.5.4 部品の Python クラスの変更

メガウイジェットの部品は夫々がいずれかの Python クラスのインスタンスである。各部品のデフォルトのクラスはリファレンス・マニュアルに記述されている。特別な *pyclass* オプションを使うことで、デフォルトとは異なったクラスを部品に結びつけることができる。例を挙げてみよう。通常 Pmw.Counter メガウイジェットは Tkinter.Label をラベルを表示するための部品として使っている。このラベルを表示するためのウイジェットとして Tkinter.Button をつかった Pmw.Counter ウイジェットを作成するには、

```
counter4 = Pmw.Counter(
        labelpos = 'w',
         label_text = 'Hello',
         label_pyclass = Tkinter.Button
)
```

とすれば良いのである。

3.6 メソッドの forwarding

Pmw メガウイジェットは正規の Python のクラスであるから、ベースのクラスから継承したメソッドとそのクラスの中で定義されたメソッドの双方を持っていてよい。Pmw はメガウイジェットが新たなメソッドを獲得する第三の方法を提供する。其の方法は一つあるいは複数の部品ウイジェットのすべてのメソッドを Forwarding する方法である。例えば、Pmw.Counter の entryfield 部品の Pmw.EntryField としての method はすべて Pmw.Counter に Forward されている。setentry() メソッドは Pmw.Counter の直接の method では無いがこの Forwarding によって、例で示した counter2.setentry() といった呼び出しが可能になっている。sententry() メソッドは Pmw.EntryField のメソッドであるので、Pmw.Counter に Forward されてこの呼び出しが可能になる。

メガウイジェットクラスやそのベースとなったクラスで定義されていないメソッド岳がForwardingの対象となる。例えば、Pmw.EntryFieldクラスは*cget()*メソッドをもっているが、このメソッドはPmw.Counterクラスには Forward されない。なぜなら、Pmw.Counterクラスはベースクラスから継承した*cget()*を既にもって入からである。

3.7 Pmw メガウイジェットの拡張

Pmw メガウイジェットを拡張するには幾つかの方法がある。まず、オプションと部品の柔軟性はウイジェットの見かけと振る舞いを大きく変更することを可能にしている。次に、interior() メソッドによってユーザの選んだウイジェットを Pmw ウイジェットの内部に埋め込むことが可能である。Pmw のクラス、MegaToplevel, MegaWidget, Dialog およびLabeledWidget はこの拡張の方法のために特に設計されている。例として、カウンタを含むダイアログを作成してみよう。

```
dialog = Pmw.Dialog(
         title = 'Counter dialog',
         buttons = ('OK', 'Cancel'))
interior = dialog.interior()
counter = Pmw.Counter(interior)
counter.pack(padx = 20, pady = 20)
```

第三の方法はメガウイジェットクラスのサブクラスを定義することである。この方法の詳細は、"How to build Pmw megawidget"の章で議論される。

3.8 簡単な例題

以下に示すのは、Pmw メガウイジェットの使用法を示す短いプログラムである。これはユーザが値を入力するのに三つの異なる方法、増減ボタンをもつカウンタ、値の検証を行う直接入力のエントリ・フィールドそして値の候補をドロップダウンのメニュで表示するコンボ・ボックス、を提供する完結したアプリケーションである。

```
import Pmw
root = Pmw.initialise(fontScheme = 'pmw1')

counter = Pmw.Counter(
        label_text = 'Counter:',
         labelpos = 'w',
         entryfield_value = '00:00:00',
         entryfield_validate = 'time',
     datatype='time',
         increment=5*60,
)
counter.pack(fill = 'x', padx = 10, pady = 10)

entry = Pmw.EntryField(
```

```
        label_text = 'Real entry:',
        labelpos = 'w',
        value = '+2.9979e+8',
        validate = 'real',
)
entry.pack(fill = 'x', padx = 10, pady = 10)

combo = Pmw.ComboBox(
        label_text = 'ComboBox:',
        labelpos = 'w',
        scrolledlist_items = map(str, range(20))
)
combo.pack(fill = 'x', padx = 10, pady = 10)

# Make the labels line up neatly
Pmw.alignlabels((counter, entry, combo))

root.title('Pmw megawidgets example')
root.mainloop()
```

3.9 別の簡単な例題

別の Pmw アプリケーションの例を示す。今度のアプリケーションは RadioSelect メガウイジェットと exit ボタンを root ウィンドウの中にもっている。

```
import Tkinter
import Pmw

def callback(tag):
     # This is called whenever the user clicks on a
     # button in the RadioSelect widget.
     print tag, 'was pressed.'

# Initialise Tkinter and Pmw.
root = Pmw.initialise(fontScheme = 'pmw1')
root.title('Pmw RadioSelect demonstration')

# Create and pack a RadioSelect widget.
radio = Pmw.RadioSelect(
        command = callback,
        labelpos = 'w',
        label_text = 'Food group:')
radio.pack(padx = 20, pady = 20)
```

```
# Add some buttons to the RadioSelect.
for text in ('Fruit', 'Vegetables', 'Cereals', 'Legumes'):
     radio.add(text)
radio.invoke('Vegetables')

# Create an exit button.
exit = Tkinter.Button(text = 'Exit', command = root.destroy)
exit.pack(pady = 20)

# Let's go.
root.mainloop()
```

3.10 Tk オプションの利用

Pmw をつかったアプリケーションをカスタマイズするのに、Tkinter のオプションデータベースをつかういくつかの方法がある。まず第一に、Tkinter の基本ウイジェットは通常の方法でカスタマイズ出来る。例えば、すべての Tkinter.Label のバックグラウンドの色を（メガウイジェットの部品であるかどうにかかかわらず）変えるには、

```
root.option_add('*Label.background', 'pink')
```

とする。一方、すべての Pmw.EntryField の *label* 部品のバックグラウンドだけを変更するには、

```
root.option_add('*EntryField.Label.background', 'green')
```

とすればよい。また、hull 部品を含むすべての Pmw.EntryField 部品のバックグラウンドを変更するには、

```
root.option_add('*EntryField*background', 'blue')
```

とすればい。これらの設定は基本的な Tk のウイジェットに影響する。そしてこの機能は Tk ウイジェットに組み込まれているので、常に利用可能である。しかし Tk オプションデータベースを Pmw メガウイジェットのデフォルト値の設定変更に使うためには、Pmw.initialise() が useTkOptionDb=1 を引数にしてよばれている必要がある。もしこのオプションが設定されていなければ Pmw ウイジェットはデフォルト値の設定に Tk オプションデータベースを検索しない。これがデフォルトの振る舞いである、これは Tk オプションデータベースを利用した際の若干の実行効率の低下を避けるためである。

useTkOptionDb オプションが設定されているとすると、すべての Pmw.Dialog メガウイジェットのデフォルトの buttonbox の位置は、

```
root.option_add('*Dialog.buttonboxpos', 'e')
```

で変更出来る。

Pmw.EntryField メガウイジェットのラベル位置を変更するために、label 部品のデフォルト値をあたえるには、

```
root.option_add('*EntryField.labelpos', 'w')
```

とすればよい。

4 Pmwのデモとテスト

Pmw は豊富なデモとテストプログラムとともに配付されている。デモ・プログラムは、Pmw でどのようなことが可能であるかの感触をつかむのに使える。またデモ・プログラムは Pmw 使用の例題として読むこともできる。テスト・プログラムはあなたの環境で Pmw を使うことに問題が無いかをチェックするために使える。

4.1 デモ・プログラム

Pmw の配付キットは Pmw メガウイジェットのほとんどの機能を見せるデモ・アプリケーションを含んでいる。デモプログラムは Pmw の各バージョンがインストールされたディレクトリのしたの demos ディレクトリに収められている。すべてのデモを含む包括的なパッケージをみるには、Python のスクリプト

```
demos/All.py
```

を実行してみればよい。

夫々のデモ・プログラムを個別に起動することもできる。ほとんどのデモ・プログラムはあるメガウイジェットの幾つかの機能を紹介する。例えば、ButtonBox メガウイジェットのデモを見るためには、demos/のあるディレクトリを/Pmw_directory としたとき、

```
/Pmw_directory/demos/ButtonBox.py
```

を実行する。

その他には以下のデモプログラムが用意されている。

BltGraph.py BLT のグラフとベクトルコマンドへの使用例

Colors.py 色のスキームの設定法

ConfigClass.py Pmw メガウイジェット部品のクラスとしての設定法。

ErrorHandling.py how Pmw displays run time errors in a window

ExampleDemo.py template for new demonstrations

Grid.py the Tkinter Grid geometry manager

LogicalFont.py how to use standard values for fonts

MessageInfo.py how to extend the Pmw MegaToplevel class

NestedDialogs.py how nested modal dialogs behave

Resources.py how to use the option database to modify Tk widget option defaults

Resources_Pmw.py how to use the option database to modify megawidget option defaults

ShowBusy.py demonstrates the Pmw interface to the BLT busy command

SpecialEntry.py deriving from Pmw.EntryField

Spectrum.py some of the Pmw color handling functions

SpeedTest.py tests the speed of creating Pmw megawidgets

TextDisplay.py how to extend the Pmw MegaWidget class

WidgetDestroy.py megawidget destruction

4.1.1 新規のメガウイジェットにかんするデモ・プログラムの作成

新規にメガウイジェットを作成したなら、ExampleDemo.py をテンプレートとして新しいメガウイジェットの機能を示すデモ・プログラムを作成しよう。

4.2 テスト・プログラム

Pmw の tests ディレクトリにはテストのためのフレームワークと一そろいの Pmw テストスクリプトが収められている。テストプログラムは標準的の Tkinter モジュールとほとんどの Pmw メガウイジェットをテストする。テストはまた、柔軟性にとんだメガウイジェットのよい見本でもある。テストを実行するには、Pmw の tests ディレクトリで、

```
python All.py
```

を実行するだけである。すべてのテストにパスすれば標準出力にはなにも出力されないはずである。もしテストに失敗することがあれば、テストの出力を Pmw の保守を担当している gregm@iname.com に送ってほしい。

すべてのテストは個別に実行することも出来る。ほとんどのテスト・プログラムはあるメガウイジェットの幾つかの機能をテストする。例えば、ButtonBox メガウイジェットをテストするためには、、

```
python ButtonBox_test.py
```

を実行する。

ファイル Test.py はテストのためのフレームワークを提供する汎用の関数を定義している。すべてのテストプログラムで Tes.py が import されている。その他のテストプログラムには、

Blt_test.py BLT vector and graph interface

Colors_test.py setting color schemes

MegaWidget_test.py creation of megawidget classes

Options_test.py option and component handling

PmwBase_test.py more option and component handling

Tkinter_test.py Tk widgets in the Tkinter module

等がある。

4.2.1 新規のメガウイジェットに対するテストプログラムの作成

新規にメガウイジェットを作成したなら、テストプログラムを作成しよう。テストプログラムの作成のためのテンプレートは用意されていないが、既存のテストプログラム（例えば scrolledText_test.pyy）を参考にすればどのようにしてテストプログラムを作成すればよいかの見通しが得られるであろう。

5 Dynamic loader: 動的ローダ

There are two aspects of Pmw, unrelated to megawidgets, that require special attention. Firstly, Pmw is made up of many sub-modules, potentially making access to its various classes and functions cumbersome for the user. Secondly, Pmw is regularly being modified and added to, thus requiring the release of new versions. Therefore, techniques for making access to the sub-modules easy and efficient and for dealing with the different versions have been developed. These techniques are incorporated into the dynamic loader which Pmw creates when it is first imported.

メガウィジェットとは関係しないが、特別に注意を払わなければならない事が二つ Pmw にはある。まず、Pmw はいくつものサブ・モジュールからでき上がっており、それらの夫々のクラスや関数へのアクセスはユーザにとって厄介なものになる可能性がある。次に、Pmw は頻繁に更新・変更されており、新しいバージョンでの置き換えがたびたび必要となる。これらの問題にたいして、サブ・モジュールへのアクセスを簡単にし、異なるバージョンの取り扱いを可能にする方法が開発された。この手法は Pmw が最初に import された際に構築されるダイナミック・ローダに統合されている。

The first purpose of the loader is to give access to all Pmw classes and functions through a single entry point, the Pmw. prefix. For example, to access the ComboBox class (which resides in one of the sub-modules of Pmw), you just have to use Pmw.ComboBox. Without the loader, this would be a more complicated reference, such as, hypothetically, Pmw.PmwComboBox.ComboBox.

ダイナミック・ローダの第一の目的はすべての Pmw クラスと関数に一つのエントリポイント Pmw 接頭詞を与えることである。たとえば、ComboBox クラス（このクラスは Pmw のサブ・モジュールの一つで定義されている）にアクセスするには、Pmw.ComboBox を使えばよい。ダイナミック・ローダがないと、このクラスへのあくせすは、例えば、Pmw.PmwComboBox.ComboBox の様な形が必要になる。

The second purpose of the loader is to delay the importing of the sub-modules until they are needed. This improves the startup time of applications which only use a few Pmw megawidgets. It also allows more megawidgets to be added to the library without slowing down applications which do not use them.

ダイナミック・ローダの第二の目的はサブ・モジュールの import を必要な時まで遅らせる事である。これによって、数個のメガウィジェットを使うだけのアプリケーションの起動時間が改善される。これはまた、新しいウィジェットの導入によって、それらのウィジェットを使うことのないアプリケーションを遅くすることなく、これらのウィジェットをライブラリに導入することができることを意味する。

The third purpose of the loader is to allow a script using Pmw to specify which version of

Pmw it requires. This allows an application to continue working correctly even after newer releases of Pmw have been made which are not compatible with the version expected by the application. Several versions of Pmw can be installed at once, with the actual version used being specified by each application. In addition, the loader can be configured to search in one or more alpha versions of Pmw. These versions may contain new megawidgets, or new versions of existing megawidgets, that are currently not in the base releases.

ダイナミックローダの第三の目的は Pmw を利用するスクリプトが利用する Pmw のバージョンを指定できるようにすることである。これによって、アプリケーションが使用している Pmw と互換性のない新しいバージョンをインストールした後も、このアプリケーションが正しいバージョンの Pmw を利用して動作しつづける事を可能にする。夫々のアプリケーションが明示的にバージョンを指定することで、同時に複数の異なるバージョンの Pmw をインストールすることが可能である。さらに、ダイナミック・ローダを設定することで、複数のアルファバージョンを検索するようにすることができる。これらのアルファバージョンはベース・リリースに含まれていない新しいメガウィジェットや既存のメガウィジェットの新しいバージョンを含んでいてよい。

Several functions are available to set and query the version of Pmw being used. These are Pmw.setversion() and Pmw.setalphaversions() which specify the version and alpha versions (if any) to use for this session; Pmw.version() which returns the version(s) being used by this session; and Pmw.installedversions() which returns the version(s) of Pmw currently installed. These are described in the Pmw functions reference manual.

Pmw のバージョンを設定したり、調べたりするための関数が用意されている。それらの関数は、そのセッションで使う Pmw のバージョンおよび（もし必要があれば）アルファ・バージョンを指定するための Pmw.setversion() および　Pmw.setalphaversions() 、現在インストールされている Pmw のバージョンのリストを値として返す Pmw.installedversions() である。これらの関数は Pmw 関数参照マニュアル Pmw functions reference manual. に記述されている。.

When Pmw is first imported, an instance of PmwLoader is created and placed into sys.modules['Pmw']. From that point on, any reference to attributes of the Pmw 'module' is handled by the loader. The real Pmw package is stored in sys.modules['_Pmw'].

Pmw が最初に import されると、PmwLoader のインスタンスが生成され、sys.modules["Pmw"] に割り当てられる。これより後では、Pmw "モジュール" の属性への参照はこのローダで処理される。じっさいの Pmw パッケージは、sys.modules['_Pmw'] に割り当てられる。

The loader searches the Pmw package base directory for sub-directories with the prefixes Pmw_ and Alpha_, which contain Pmw base releases and alpha releases. The version numbers are given by the part of the directory name following the prefix. These versions are available for use and are those returned by the Pmw.installedversions function. The initial version is set to the base release with the greatest version number. When the first reference to a Pmw class or function is made, the loader reads the files named Pmw.def in the current base version directory and also in the alpha directories (if any). These files list all the classes and functions supported by the version. Pmw attributes are first searched for in the alpha directories and then in the base version directory. The first directory which supports the reference is used. In this way, alpha versions override base versions.

ローダは Pmw パッケージのベース・ディレクトリを Pmw_および Alpha_ではじまる名前を持つサブ・ディレクトリから探しだす。Pmw_および Alpha_ではじまる名前を持つサ

ブ・ディレクトリは夫々ベース・リリースおよびアルファリリースを含んでいる。バージョン番号は接頭詞 (Pmw_あるいは Alpha_) に続く名前で示される。これらのバージョンは後で利用できるまたこれらのバージョン番号は Pmw.installedversions () 関数の戻り値として返される。バージョンの初期値はそれらのインストールされたバージョンのうち最大のバージョン番号をもったリリースデある。最初に Pmw のクラスあるいは関数にアクセスがあった時にローダは、現在のベースバージョンのディレクトリおよびアルファ・ディレクトリにある Pmw.def という名前のファイルを読み込む。これらのファイルにはそのバージョンで利用可能なすべてのクラスと関数のリストが記述されている。Pmw の属性はまずアルファ・バージョンのディレクトリから検索され、その後にベース・ディレクトリが検索される。最初にこの属性 (クラス、関数、サブモジュール）をサポートするディレクトリが使われる。このようにしてアルファバージョンがベースバージョンをオーバーライドする。

The directory Alpha_99_9_example contains a simple example of how to structure an alpha version. The following code can be used to request that the alpha version be used and then creates an instance of a new megawidget defined in the alpha version.

ディレクトリ Alpha_99_9_example にはアルファ・バージョンの構造をしめすための簡単な例が収められている。次に示すコードは使用するアルファバージョンを指定し、そのアルファバージョンで定義されている新たなメガ・ウィジェットのインスタンスを作成する。

```
import Pmw
Pmw.setalphaversions('99.9.example')

# ベース Pmw バージョンを使って標準のメッセージ・ダイアログを作成する。
ordinary = Pmw.MessageDialog(
    message_text = 'Ordinary\nPmw Dialog')

# アルファ Pmw バージョンを使ってサンプルダイアログを作成する。

alpha = Pmw.AlphaExample()
```

5.1 Freezing Pmw

Since the dynamic loader requires that Pmw be installed at run time, it can not be used when freezing Pmw. In this case, a single module containing all Pmw code is required, which can then be frozen with the rest of the application's modules. The bundlepmw.py script in the Pmw bin directory can be used to create such a file. This script concatenates (almost) all Pmw megawidget files into a single file, Pmw.py, which it writes to the current directory. The script is called like this:

```
bundlepmw.py [-noblt] [-nocolor] /path/to/Pmw/Pmw_X_X_X/lib
```

The last argument should be the path to the lib directory of the required version of Pmw. By default, the Pmw.py file imports the PmwBlt and PmwColor modules and

so, to freeze an application using Pmw, you will need to copy the files PmwBlt.py and PmwColor.py to the application directory before freezing.

If you are sure that your application does not use any of the Pmw.Blt or Pmw.Color functions, you can use the -noblt or -nocolor options. In this case Pmw.py will be modified so that it does not import these module(s) and so will not need to be included when freezing the application.

If your application only uses a few Pmw megawidgets, you can remove the references to the ususued ones in the files list in the bundlepmw.py code. To make the change, take a copy of the script and modify it. This will make the Pmw.py file smaller. However, be sure that you do not delete megawidgets that are components or base classes of megawidgets that you use.

6 参照マニュアル

6.1 基本クラス

6.1.1 Pmw.MegaArchetype

名前 Pmw.MegaArchetype() - abstract base class for all Pmw megawidgets

機能 このクラスはすべての Pmw メガウイジェットの基本クラスとなる。このクラスはオプションと部品ウイジェットを取り扱うメソッドを提供する。このクラスは通常他のクラスの基本クラスとなることで使用される。Pmw.MegaWidget クラスや Pmw.MegaToplevel の様に hullClass が指定されていればコンテナ・ウイジェットがすべての部品ウイジェットの親となるように生成される。これらのクラス（Pmw.MegaWidget クラスや Pmw.MegaToplevel）を継承するサブ・クラスはその他の部品ウイジェットやオプションを提供することで、アプリケーション中で使用可能なメガウイジェットを実装する。

hullClass 引数が生成子に与えられないと、コンテナ・ウイジェットは生成されず、オプションの設定機能だけが有効となる。

部品ウイジェット (Component) 大体において、メガウイジェットはコンテナ・ウイジェットに詰め込まれた他のメガウイジェットから構成されている。

それらの副ウイジェットはメガウイジェットの部品とよばれる。これらの部品には参照を容易にするためにそれぞれ名前がつけられている。メガウイジェットの部品の機構はメガウイジェットの内部にユーザが立ち入ることを可能にしている。たとえば、各部品のメソッドをユーザのプログラムが利用したり、それらの部品の設定オプションをつかったりできる。

副部品 sub componetnts 部品がまた副部品を含むメガウイジェットであることもある。この場合、これらの副部品は、component_subcomponent の形式の名前で引用することができる。たとえば、Pmw.ComboBox は entryfield という名前の部品をもている。部品 entryfield は Pmw.EntryField のインスタンス (実体) である。そして、Pmw.EntryField は Tkinter.Entry を部品 entry として持っている。combobox のコンテキストでは、この

Tkinter.Entry ウイジェットのインスタンスは entryfield_entry という名前でアクセスできる。

部品の別名　副部品の名前は前節の規約に従うと、不便なほど長くなってしまうことがある。これを避けるために部品や副部品には別名 (aliase) をつけることができる。たとえば、Pmw.ComboBox の entryfield_entry 副部品は、簡単に、entry と呼ぶことができる。名前の衝突を生じないかぎり、副部品の名前はそのまま部品の名前と同じように使える。

部品のグループ化　メガウイジェットの似たようなコンポーネントにはグループ名をあたえることができる。グループ名の利用によってグループ内のすべての部品にグループ名でアクセスできる。例えば、Pmw.Counter の二つの矢印部品には Arrow というグループ名が与えられている。また Pmw.ButtonBox のように、いくつでも同じ部品をつくりだすことのできるメガウイジェットでは、それらの部品をグループ名を使って作りだすことができる。規約では、グループ名は大文字で始まることになっている。

オプション　メガウイジェットの見栄えや振るまいをユーザが変更できるように、メガウイジェットはオプションを持っている。Tkinter のウイジェットと同じ技法をつかって、メガウイジェットのオプションの値は、メガウイジェットが生成されるときの初期化関数と configure() 関数の引数として与える。オプションの現在値は cget() あるいは configure() メソッドで読み出される。Tkinter ウイジェットと同様にメガウイジェットのオプションはデフォールト値に初期化される。もし、Pmw.initialise() のオプションである useTkOptionDb が設定されているときには、メガウイジェットのオプションの初期値を設定するために、Tk のオプションデータベースが検索される。Tk のオプションデータベース中から引きだされた値は文字列から Python のオブジェクトに変換される。変換には制限付きの eval() がつかわれる。eval() で評価できない値はすべて文字列として取り扱われる。

Inherited options　クラスの中で定義されているオプションと同様に、誘導されたクラスは親クラスとおなじメンバを継承する。親クラスで定義されている各オプションのデフォルト値は小クラスの中で変更してよい。

初期化オプション：メガウイジェットのオプションには生成子の呼び出しの中でのみ設定可能なものがある。通常のオプションとは異なり、これらの初期化オプションを設定するために configure() 関数の呼び出しを利用することはできない。

部品オプション：メガウイジェット部品のオプションは記法 component_option で参照することができる。メガウイジェットそのもののオプションと同じく、部品のオプションは生成子呼び出し, cget() 呼び出し、 および configure() 呼び出しで使用される。例えば、Pmw.ScrolledText のもつ部品 text は Tkinter.Text ウイジェットとして state オプションを持つが、このオプションは、

```
widget.configure(text\_state = 'disabled')
```

の様に使われる。

副部品, 部品の別名、部品グループもオプションと組み合わせて使われる。一例を挙げると、Pmw.ComboBox メガウイジェットの部品である entryfield_entry の state オプションは、

```
combobox.configure(entryfield\_entry\_state = 'normal')
```

によって設定できる。この部品は別名を持っているので、等価な次の呼び出しを使うほうが便利である。

```
combobox.configure(entry\_state = 'normal')
```

別の例として、Pmw.Counter メガウイジェットの二つの矢印のバックグラウンドの色を、Arrow 部品グループを使って設定してみると次のようになる。

```
counter.configure(Arrow\_background = 'aliceblue')
```

The pyclass 部品オプション The pyclass component option は、非標準の部品としてのパイソンクラスを指定する特別な記法である。このオプションは部品が生成される時にのみ指定可能である。親メガウイジェットの生成子呼出中に生成される部品ウイジェットにたいして、このオプションは生成子の引数として、component_pyclass の形で与えられる。例えば、Pmw.TextDialog の text 副部品のクラスを FindText.Text とするには、

```
dialog = Pmw.TextDialog(text\_pyclass = FontText.Text)
```

とする。親メガウイジェットの生成子呼出しのあとに部品が生成される部品では、pyclass オプションは部品ウイジェットを生成するメソッドに引数として与えられる。例えばつぎの呼び出しは、Pmw.Button の button オブジェクトのクラスを MyButton クラスに変更するには、

```
buttonBox.add('special', pyclass = MyButton)
```

とする。

新しい部品となる python クラスはメガウイジェットが部品に適応するすべての method とオプションをサポートせねばならない。それぞれの部品に必要なインタフェイスは文書化されていない。新しい python クラスを使うためには、Pmw のソースコードをみる必要があるだろう。それぞれの部品のデフォルト・クラスから導出されたクラスであれば、親クラスの method とオプションをサポートしているかぎりにおいて、部品のクラスを置き換えることができるだろう。

例えば、Tkinter.Text から導出されたクラスはどれであれ、Pmw.TextDialog の text 副部品として使うことができる。

pyclass 部品オプションはいくつかの Tkwidget でサポートされている class オプションと混同してはならない。class オプションは Tk オプション・データベースクラスを設定し、Tk によって Tk ウイジェットのその他のオプションのデフォルトの値を調べるデータベースとして Tk によって使用される。pyclass という名前は既存のどの Tk オプションとも衝突が起きないように選ばれた。

メガ・ウイジェットの生成 このクラスを継承するすべてのクラスの生成子は同じ引数を持つ。すなわち、一つの位置引数と任意個数のキーワード引数である。位置引数はデフォルト値として None(root ウィンドウを意味する) をもち、メガ・ウイジェットの親となるウイジェットを指定する（より正確にはメガウイジェットの hull 部品のおやであるが)。キーワード引数はオプションの初期値を設定する。導出クラスの生成子の形式は：

```
def __init__(self, parent = None, **kw):
```

である。

メソッド

addoptions(optionDefs) 新たにオプションをこのメガウイジェットに追加する。引数 optionDefs はdefineoptions() メソッドの引数と同様に取り扱われる。

このメソッドは導出クラスでの使用を想定してつくられた。このメソッドはメガウイジェットが条件付きでいくつかのオプションを定義する必要があるときにのみ使われる。この条件とは、おそらく他のオプションのとる値に鳴だろう。通常メガウイジェットはすべてのオプションを defineoptions() の呼び出しで定義し、addoptions() メソッドを呼ぶ必要はない。このメソッド呼び出されるとすれば、defineoptions() 呼び出しの後で、かつ initialiseoptions() 呼び出しの前である。

cget(option) オプションの現在値を返す。返り値の形式はオプションの節で説明された形式である。このメソッドは、オブジェクトの subscript 形式、例：myWidget['font']、でも使える。常に値として文字列を返す Tkinter の cget() とは異なり、このメソッドはオプションがセットされた時のデータ型と値を返す。例外はオプションが部品のオプションであり、この部品が Tkinter のウイジェットであるときだけである。このときには、このメソッドは Tcl/Tk が返す文字列を値として返す。

6.1.2 Pmw.MegaArchetype

名前 Pmw.MegaWidget() - フレームの中に収まるメガウイジェットの基本となるクラス。

親クラス Pmw.MegaArchetype

記述 このクラスは Tkinter.Frame にすっぽり含まれてしまうメガウイジェットを生成する。このクラスは自身が Toplevel ウィンドウとなるのではないウイジェットの親クラスになる。Pmw.ButtonBoc, pme.comboBox はその様なウイジェットである。

部品 このメガウイジェットとその基底クラスによって生成される部品を説明する。

hull この部品はメガウイジェット全体の本体として振る舞う。そのたの部品はこの hull の子供としてつくられる。デフォルトではこの部品は Tkinter.Frame である。

メソッド このメガウイジェットに特有なメソッドだけをいかに記述する。親クラスから継承されたメソッドについては、それらの基底クラスのマニュアルを参照のこと。さらに、Tkinter.Frame のメソッドは hull 部品に継承されている。

destroy hull 部品とその子供ウイジェットを消滅させる。

6.1.3 Pmw.MegaToplevel

名前 Pmw.MegaToplevel() - Toplevel ウィンドウとなるメガウイジェットの基底クラス。

親クラス Pmw.MegaArchetype

概要 このクラスは独立ナウィンドウをもつメガウイジェットを作成可能とする。あるいはこのクラスは、Pmw.Dialog の様に、より特別かしたトップレベルのメガウイジェットをつくる基底クラスとなる。このクラスでは、hull とよばれる Tkinter.Toplevel 部品がつくられる。hull はめメガウイジェットのコンテナとして働く。hull ウイジェットの Window クラス名はメガウイジェットをもっともよく表すクラス名がセットされる。このクラスからの導出クラスでは、hull ウイジェットの子供としてその他のウイジェット部品を生成することで、新たな機能を導入する。

このメガウィジぇトは通常の Toplevel ウィンドウとして、またモーダルなダイアログウィンドウとしても使われる。通常の使用では show() および withdraw() メソッドが使われ、モーダルダイアログとして使われるときには、activate() および deactivate() メソッドが使われる。もしも通常の Toplevel ウィンドウとして表示されているときに、Window Manager によって削除されたときの標準の振る舞いはウィンドウを破棄することである。モーダルダイアログとしての動作中にウィンドウが削除されたときには単に deactivate() される。 userdeletefunc() および usermodaldeletefunc() メソッドを使ってこれらの振る舞いを変更できる。もしこのウィンドウをモーダルダイアログとして使うなら、WM_DELETE_WINDOW ウィンドウマネージャプロトコルを設定するために、protocol() メソッドを読んではならない。

ある時点でアクティブなウィンドウはスタックを形成する。もっとも最近にアクティブ状態になったウィンドウがそのスタックの最上部に位置する。すべてのマウスエベントとキーボードエベントはこの最上部のウィンドウに送られる。このウィンドが デアクティベイトされた時には、スタックの次にあるウィンドウがこれらのエベントを受け取るようになる。

部品 このメガウィジェットとその基底クラスでつくられる部品は以下のとおり。

hull この部品はこのメガウィジェットの本体として振る舞う。その他の部品は hull の子供として生成される。デフォルトでは hull 部品は Tkinter.Toplevel である。

メソッド このメガウィジェット特有のメソッドだけについて記す。基底クラスなどから継承されたメソッドについてはそのクラスのマニュアルを見ること。基底クラスからのメソッドにくわえて、Tkinter.Toplevel メソッドが hull 部品に継承されている。

activate(globalMode = 0, master = None, geometry = 'centerscreenfirst') メガウイジェットのウィンドウを表示し、grab 操作を行い、結果を待つ。deavtivate() メソッドがよばれたときには、ウィンドウを withdraw し、grab を解放し、結果を返す。

globalMode が 0 であればポインタとキーボードの制御をつかんでしまう。これによって、アプリケーションの他のウィンドウにエベントが配達されるのを防ぐ。glovalMode が 1 であるときには、アプリケーションに関係なく自分自身以外のすべてのトップレベルウィンドウにエベントが配達されないようにする。(モーダル・ダイアログ) このような大域的なグラブの使用はできるだけひかえるべきである。

ウィンドウが表示されるとき、そのディスプレイ上の位置は geometry 引数の値によって決められる。geometory 引数の値は、以下に示される centerscreenfirst,centerscreenalways,first + spec および spec のうちの一つである。

- centerscreenfirst ウィンドウは初めてアクティブになったときディスプレイの中心に配置される。その後の active() 呼び出しでは、ユーザがウィンドウを移動させていたとしても、最後に表示されていたのと同じ位置に表示される。

- centerscreenalways ウィンドウは常にスクリーンの中心に配置される。

- first+ *spec* underlinefirst 以降の引数 *spec* は Xwindow 標準の geometry 指定と解釈される。最初に activate された時には、この指定の場所に表示される。その後の active() 呼び出しでは、ユーザがウィンドウを移動させていたとしても、最後に表示されていたのと同じ位置に表示される。例えば、geometri=first+100+100 と指定されたときには、最初に表示されるウィンドウはスクリーン上の (100,100) に配置される。その後の activate() 呼び出しは直前の位置を変えない。

- *spec* これは標準的な geometry 指定である。ウィンドウはこの指定にしたがって配置される。BLT Tcl 拡張ライブラリがある場合には、時計カーソルがウィンドウの deactivate() メソッドが呼び出されるまで表示される。acitivatecommand オプションが callable であるときには、activatecommannd が、ウィンドウが結果を待ち始める直前に実行される。その Master が None 意外のものであるときには、ウィンドウはマスタウィンドウの一時的ウィンドウとなる。このときのマスタはその他の既に存在する toplevel ウィンドウである必要がある。

active() ウィンドウがアクティブであれば（すなわち activate() は deactivate() 呼び出しによって返される値を待っている）1 を返す。 さもなければ 0 を返す。

deactivate(result = None) このメソッドは acitivate() 呼び出しが待ち状態のときに呼び出される。この呼び出しはウィンドウをスクリーンから消去し、event のグラブを解放し、activate() 呼び出しに結果の値を返す。deactivatecommand オプションが呼び出し可能であるときには、deactivate() 呼び出しが終了する直前に deactivatecommand が実行される。

destroy() hull 部品とその子供のウイジェットを破棄する。メガウイジェットが動作中 (active) であれば deactivate() を呼ぶ。

show()　ウィンドウを表示する。トップレベルウィンドウを最表面のウィンドウにするか、icon 状態からウィンドウ状態に戻す。ウィンドウがすでに表示された状態であれば、直前の位置を変えない。つまり、withdraw() を呼んだあとで、show() を呼んでもウィンドウの位置は変化しないが、withdraw() のあとで deiconify() を呼んだ場合には場所がかわうる。

userdeletefunc(func = None)　もし func が None であれば、ウィンドウがウィンドウマネージャによって削除される際、ウィンドウが表示されているあいだに呼ばれる関数を返す。func が None でなければ、その関数を設定する。デフォルトではこの関数は self.destroy である。

usermodaldeletefunc(func = None)　もし func が None であれば、ウィンドウがウィンドウマネージャによって削除される際、ウィンドウがアクティブなあいだに (つまりモーダルダイアログとして使われているとき) 呼ばれる関数を返す。func が None でなければ、その関数を設定する。デフォルトではこの関数は self.deactivate である。

6.2　ウイジェット

6.2.1　Pmw.ButtonBox

名前　Pmw.ButtonBox() -ボタンのマネージャウイジェット

親クラス　Pmw.MegaWidget

オプション

部品

メソッド

```
def _checkForBlt(window):
```

```
def haveblt(window):
def _loadBlt(window):
def busy_hold(window):
def busy_release(window):
class Vector:
    def __init__(self, size=None, master=None):
    def __del__(self):
    def __str__(self):
    def __repr__(self):
    def __cmp__(self, list):
    def __len__(self):
    def __getitem__(self, key):
    def __setitem__(self, key, value):
    def __delitem__(self, key):
    def __getslice__(self, start, end):
    def __setslice__(self, start, end, list):
    def __delslice__(self, start, end):
    def set(self, list):
    def get(self):
    def append(self, value):
    def count(self, start):
    def search(self, start, end=None):
    def index(self, value):
    def insert(self, index, value):
    def remove(self, value):
    def reverse(self):
    def sort(self, *args):
    def sort_reverse(self, *args):
    def min(self):
    def max(self):
    def clear(self):
    def delete(self, *args):
    def length(self, newSize=None):
    def range(self, first, last=None):

class Graph(Tkinter.Widget):
    def __init__(self, master=None, cnf={}, **kw):
    def _configure(self, subcommand, option, kw):
    def extents(self, item):
    def invtransform(self, winX, winY):
    def transform(self, x, y):
    def axis_cget(self, axis, key):
    def axis_configure(self, axis, option=None, **kw):
    def axis_invtransform(self, axis, value):
```

```
def axis_limits(self, axis):
def axis_transform(self, axis, value):
def xaxis_cget(self, key):
def xaxis_configure(self, option=None, **kw):
def xaxis_invtransform(self, value):
def xaxis_limits(self):
def xaxis_transform(self, value):
def x2axis_cget(self, key):
def x2axis_configure(self, option=None, **kw):
def x2axis_invtransform(self, value):
def x2axis_limits(self):
def x2axis_transform(self, value):
def yaxis_cget(self, key):
def yaxis_configure(self, option=None, **kw):
def yaxis_invtransform(self, value):
def yaxis_limits(self):
def yaxis_transform(self, value):
def y2axis_cget(self, key):
def y2axis_configure(self, option=None, **kw):
def y2axis_invtransform(self, value):
def y2axis_limits(self):
def y2axis_transform(self, value):
def crosshairs_cget(self, key):
def crosshairs_configure(self, option=None, **kw):
def crosshairs_off(self):
def crosshairs_on(self):
def crosshairs_toggle(self):
def element_activate(self, name, *args):
def element_bar(self, name, **kw):
def element_cget(self, name, key):
def element_closest(self, x, y, *args, **kw):
def element_configure(self, name, option=None, **kw):
def element_create(self, name, **kw):
def element_deactivate(self, *args):
def element_delete(self, *args):
def element_exists(self, name):
def element_line(self, name, **kw):
def element_names(self):
def element_show(self, nameList=None):
def element_type(self, name):
def grid_cget(self, key):
def grid_configure(self, option=None, **kw):
def grid_off(self):
def grid_on(self):
```

```
def grid_toggle(self):
def legend_activate(self, *args):
def legend_cget(self, key):
def legend_configure(self, option=None, **kw):
def legend_deactivate(self, *args):
def legend_get(self, pos):
def postscript_cget(self, key):
def postscript_configure(self, option=None, **kw):
def postscript_output(self, fileName=None, **kw):
def marker_after(self, first, second=None):
def marker_before(self, first, second=None):
def marker_cget(self, name, key):
def marker_configure(self, name, option=None, **kw):
def marker_create(self, type, **kw):
def marker_delete(self, *args):
def marker_exists(self, name):
def marker_names(self, pattern=None):
def marker_type(self, name):
```

8 メガウイジェットのソースコード

〈省略〉

9 バージョンの異なるPmwの間でのアプリケーションの移植

This document contains a brief guide to porting existing code between different versions of Pmw. It includes significant functionality changes but does not include bug fixes or compatible enhancements. For details of all changes, see Changes to Pmw versions.

この文書は既にある Pmw を使う Python プログラムを異なるバージョンの Pmw へ移植する際の簡単なガイドである。この文書では主要な機能の変更については述べているが、バグフィックスや互換な機能の増強については触れていない。すべての変更のしょうさいについては、Changes to Pmw versions. を参照されたい。

9.1 Porting from 0.8.3 to 0.8.4（0.8.3 から 0.8.4 への移植）

Change the setnaturalpagesize() method of Pmw.NoteBook to setnaturalsize() (to be consistent with Pmw.PanedWidget).

Pmw.NoteBootk の etnaturalpagesize() メソッドを setnaturalsize() に変更のこと。(これにより、Pmw.PangeWidget() と統一性が保たれる。)

Change Pmw.excludefrombusycursor() to Pmw.setbusycursorattributes(). Replace busyCursorName option of Pmw.initialise() with cursorName attribute of Pmw.setbusycursorattributes().

Pmw.excludefrombusycursor() メソッドを Pmw.setbusycursorattributes() に変更しなさい。Pmw.initialise() の busyCursorName オプションを Pmw.setbusycursorattributes() の cursorName 属性に変更しなさい。

Several rarely used key bindings for Pmw.ScrolledListBox were removed, changing the behaviour of the megawidget.

めったに使われることの無い、Pmw.ScrolledListBox のキーバインドの幾つかは取り除かれた。

9.2 Porting from 0.8.1 to 0.8.3（0.8.1 から 0.8.3 への移植）

The megawidgets Pmw.NoteBookR and Pmw.NoteBookS have been replaced by a new Pmw.NoteBook. The interfaces are not compatible, so see the Pmw.NoteBook reference manual for details.

メガウィジェット Pmw.NoteBookR と Pmw.NoteBookS は Pmw.NoteBook で置き換えられた。インタフェースは古いウィジェットと互換ではない、詳細は Pmw.NoteBook リファレンスマニュアルを参照のこと。

Change the get() method of Pmw.OptionMenu to getcurselection() and the remove() method of Pmw.PanedWidget to delete(). Pmw.OptionMenu ウィジェットの get() メソッドは getcurselection() で、また Pmw.PanedWidget の remove() メソッドは delete() で置き換えられた。

If you use 'end', 'default' or None in calls to the index() method of several megawidgets, change these to Pmw.END, Pmw.DEFAULT and Pmw.SELECT, respectively.

幾つかのメガウィジェットの index() メソッド中で end', 'default' あるいは None を使っているなら、それらを Pmw.END, Pmw.DEFAULT および Pmw.SELECT, で置き換える。

The exclude argument has been removed from Pmw.showbusycursor(). Use Pmw.excludefrombusycu
instead. Pmw.showbusycursor() の exclude 引数は取り除かれた。かわりに、Pmw.excludefrombusycursor
を使う。

The names of some of the positional arguments in the following methods have changed:
MegaArchetype.createcomponent(), ButtonBox.insert(), ButtonBox.add(), MenuBar.addcascademenu(
MenuBar.addmenuitem() and RadioSelect.add().

以下のメソッド中の位置指定による引数の名前の幾つかが変更された。:MegaArchetype.createcompone
ButtonBox.insert(), ButtonBox.add(), MenuBar.addcascademenu(), MenuBar.addmenuitem()
および RadioSelect.add().

The Pmw.maxfontwidth() function has been removed. Use the font_measure() Tkinter method, or if that has not yet been implemented:

Pmw.maxfontwidth() 関数は取り除かれた。 Tkinter の font_measure() メソッドをかわりに使う。もしこのメソッドが実装されていなければ、代わりに以下の方法を用いる。

```
someWidget.tk.call('font', 'measure', someFont, 'W')
```

The Pmw.fontexists() function has been removed. This is because, since Tk8.0, all fonts exist, so it no longer has any meaning.

Pmw.fontexists() 関数はもはや存在しない。Tk8. 以降ではすべてのフォントが存在するためこの関数が無意味になったためである。

9.3 Porting from 0.8 to 0.8.1（0.8 から 0.8.1 への移植）

The Blt.Graph now supports blt2.4i which is not backwards compatible with blt2.1.

Blt.Graph は、blt2.1 とは互換ではないところの blt2.4i をサポートする。

9.4 Porting from 0.7 to 0.8（0.7 から 0.8 への移植）

The format argument of Pmw.datestringtojdn() now defaults to 'ymd'. If you want to display dates with year, month and day in a different order, add a format option to Pmw.datestringtojdn() or to the datatype option of Pmw.Counter or the validate option of Pmw.EntryField.

Pmw.datestringtojdn() の format 引数の既定値は'ymd' となった。もし年、月、日の順序が異なった表示を必要なら、format オプションを Pmw.datestringtojdn() に追加するかあるいは、Pmw.Counter の datatype オプションを使う、または Pmw.EntryField の validate オプションを使う。

The justify() method from Pmw.ScrolledListBox has been removed. Use the xview() or yview() methods instead.

Pmw.ScrolledListBox ウィジェットから justif() メソッドが取り除かれた。代わりに xview() メソッドおよび yview() メソッドを使う。

Replace the getFrame() method of Pmw.ScrolledFrame with the interior() method.

Pmw.ScrolledFrame ウィジェットの getFrame() メソッドを interior() メソッドで置き換える。

Replace the ringpadx and ringpady options of Pmw.Group by padding the megawidget itself or by padding the children of the megawidget.

Pmw.Group ウィジェットの ringpadx and ringpady オプションはメガウィジェット自体の padding あるいはメガウィジェットの子供ウィジェットの padding で置き換える。

Replace the canvasbind() and canvasunbind() methods of Pmw.Balloon with tagbind() and tagunbind().

Pmw.Balloon ウィジェットの canvasbind() および canvasunbind() メソッドは tagbind() and tagunbind() で置き換える。

The return value of Pmw.EntryField command callback is now ignored. Previously, if the callback destroyed the megawidget, it was required to return the string 'break', to work around a problem in the event handling mechanism in Tkinter. With python 1.5.2, Tkinter has been fixed. Therefore, user-supplied callback functions should use Pmw.hulldestroyed to check if the megawidget has been destroyed before performing any operations on it.

Pmw.EntryField ウィジェットの command にしていされたコールバック関数の戻り値は無視されるようになった。以前はコールバック関数がメガウィジェットを破壊した場合には、コールバック関数は値として文字列'break' を返すように要求されていた。これは Tkinter のエベント処理機構の問題を避けるために必要であった。Python1.5.2 でこの Tkinter の問題は修復された。従ってユーザ定義のコールバック関数はメガウィジェットに対する操作を行う前に、メガウィジェットが存在するかどうかを Pmw.hulldestroyed を用いてチェックすべきである。

If you require the 'pmw1' fontScheme when running under Microsoft Windows and Macintosh, you will need to set the Tk font options manually.

Microsoft Windows および Macintosh 上で Pmw を走らせる際に 'pmw1' フォント・スキムが必要であるなら、Tk font オプションを別途設定する必要がある。

9.5 Porting fom 0.6 to 0.7

Replace the maxwidth option of Pmw.EntryField with the 'max' field of the validate option.

To specify that there should be no validation performed for a Pmw.EntryField, the validate option must be None, not '' as before.

The date and time values of the Pmw.EntryField validate option (such as 'date_dmy' and 'time24', etc) are no longer supported. Instead use a dictionary as the value of the validate option with 'date' or 'time' in the 'validator' field. Include other fields in the dictionary to further specify the validation.

Pmw.Counter no longer supports the old date and time values for the datatype option. Use a dictionary with a 'counter' field of 'date' or 'time' and other fields to further specify the counting.

Pmw.Counter no longer supports the min and max options. Use the Pmw.EntryField validate option instead.

The bbox method of Pmw.ScrolledListBox now refers to the bbox method of the listbox component, not the hull component.

By default, Pmw.MenuBar now automatically adds hotkeys to menus and menu items for keyboard traversal. To turn this off, use the hotkeys = 0 option.

The createcomponent() method now disallows the creation of component names containing an underscore. If any component names contain an underscore, rename them.

9.6 Porting from 0.5 to 0.6

To port applications using Pmw version 0.5 to version 0.6, make sure you are using python1.5. Then, simply change any lines in your application like this:

```
from PmwLazy import Pmw
```

to this:

```
import Pmw
```

Also, if you have added the lib directory of a specific version of Pmw to sys.path or PYTHONPATH, this can be removed, as long as Pmw can now be found from the default path, such as in the python site-packages directory.

9.7 Porting from 0.2 to 0.4

To get Pmw.0.2 default fonts (helvetica with bold italic menus and italic scales) initialise with:

Pmw.initialise(fontScheme = 'pmw1')

If no fontScheme is given, the standard Tk default fonts are used.

Remove all calls to setdefaultresources(), usual(), keep(), renameoptions(), ignore() and defineoptiontypes().

Move call to defineoptions() to before call to base class constructor, create optiondefs tuple from self.defineoptions arguments, then call defineoptions().

Remove resource class and name from optiondefs.

The last element in the optiondefs tuple (callback function) must be given (may be None).

Add to classes currently without any options:

optiondefs = () self.defineoptions(kw, optiondefs)

Use createcomponent() to create components - this replaces the calls to the component widget constructor and to registercomponent().

Do not inherit from Pmw.LabeledWidget. Instead, replace with Pmw.MegaWidget with labelpos and labelmargin options and a call to self.createlabel(). If calling createlabel(), must replace pack() with grid().

When calling a megawidget constructor, include subcomponent name when setting subcomponent options (eg labeltext -¿ label_text)

The items option of ScrolledListBox is an initialisation option only - use setlist() method after initialisation.

The autorelief option for Counter, EntryField, ScrolledText, TextDialog has been removed.

ScrolledListBox.getcurselection() always returns a tuple of strings, possibly of zero length.

Counter increment is always initialised to 1.

The 'time' Counter datatype option has been replaced by 'timeN' and 'time24'.

The 'time' EntryField validate option has been replaced by 'timeN' and 'time24'.

Replace call to initialise() with initialiseoptions(), removing "kw" arg. This should always be the last line in a megawidget constructor.

Replace hide() with withdraw().

Now need iconpos option for MessageDialogs with icon_bitmap option set.

Example megawidget class definition:

```
class MyBigWidget(Pmw.MegaWidget):
    def __init__(self, parent = None, **kw):

        # Define the megawidget options.
        optiondefs = (
            ('errorbackground',   'pink',      None),
            ('maxwidth',          0,           self._myfunc),
```

```
    ('myinit',            'good',      Pmw.INITOPT),
)
self.defineoptions(kw, optiondefs)

# Initialise the base class (after defining the options).
Pmw.MegaWidget.__init__(self, parent)

# Create the components.
interior = self.interior()
self._widget = self.createcomponent('component',
        (('alias', 'component_alias'),), None,
        Tkinter.Button, (interior,))
self._widget.grid(column=0, row=0, sticky='nsew')

self.createlabel(interior)

# Initialise instance variables.
self.deriveddummy = None

# Check keywords and initialise options.
self.initialiseoptions(MyBigWidget)
```

10 Pmwのバージョン毎の違い

〈省略〉

11 謝辞

The initial ideas for Pmw were blatantly stolen from the itcl extensions [incr Tk] by Michael McLennan and [incr Widgets] by Mark Ulferts. Several of the megawidgets are direct translations from the itcl to python.

The base classes and most megawidgets were written by Greg McFarlane and Peter Munnings. The TimeCounter megawidget was contributed by Joe VanAndel. A big thank you to the following people for their bug reports, fixes, enhancements and suggestions: Joe Saltiel, Andreas Kostyrka, Case Roole, Michael McLay, Clemens Hintze, Magnus Lycka, Roman Sulzhyk and Guido van Rossum. Special thanks to Case Roole and Michael McLay for help with getting Pmw to work with python 1.5 packages and many other nifty features. My deepest apologies if I have forgotten anyone. Please let me know.

The Pmw web and ftp site is made available courtesy of Dumpleton Software Consulting Pty Limited.

The current maintainer is Greg McFarlane. If you have any comments, enhancements or new contributions, please contact me at gregm@iname.com.

A Pmw コピーライト

参考文献

[1] N. Yamamoto “gnuplot.py の使い方”, in preparation

Contents