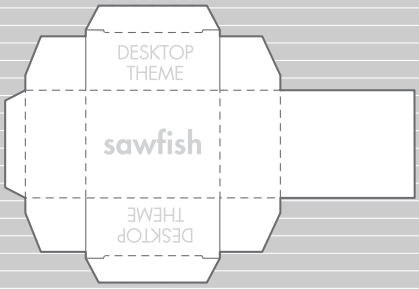


sawfishテーマのすべて

佐藤暁(ss@gnome.gr.jp)

デスクトップの模様替え

私事で恐縮ですが、最近自宅でささやかな模様替えをしました。テーブルを置き、家具の配置換えなどをし、仕事から帰った時に、よりくつろげるような快適な部屋になりました。

もともとあまりマメな方ではないのですが、急に一念発起して部屋全体を掃除、ついでに模様替えしたりすることはよくあります。特にテスト前や、仕事がたてこんでいるときなど、忙しくて時間のないときに限って部屋の大改造をしたくなったりします :-)。このような、気分転換のための部屋の模様替えは、読者の方もよくやることだと思います。その意味では、ウィンドウマネージャのテーマを作るというのは、ちょうどデスクトップの模様替えのようなものといえます。

sawfishの場合は、Enlightenment、Icewmなど他の多くのウィンドウマネージャと同じように、機能面の改造も含んだ「テーマ機能」を持っていますので、好きなだけルック&フィールを自分好みにできます。

sawfishのテーマは、sawfish themes org(以下s.t.o.。記事末のRESOURCE[1]を参照)からダウンロードした既製品をそのまま使うもよし、それをちょっと加工して自分の好みのスタイルにしてしまうもよしで、気軽にデスクトップの模様替えを行えます。あるいは気合いを入れて、ゼロから自分独自の世界を作る(完全にゼロからテーマを作る)のも可能で、デスクトップの様子を自由に替えられます。また「模様替え」の中には、単に外観を変えるだけでなく、機能の面で自分が欲する、特化した環境にしたいということも当然含まれます。

以下簡単でありますが、sawfishでテーマを作る方法を、具体的にその流れを追っていきながら紹介したいと思います。

1 sawfishとは?

みなさんは、デスクトップ環境としてGNOMEとKDEのどちらをお使いでしょうか? 中には「私は統合デスクトップ環境など使わない、twm、fvwmで十分」というストイック :) な方*1や、高機能でバランスのとれたWindow Makerをずっと気に入って使っているという方、ビジュアルについてはおそらく現時点でも最高峰の、Enlightenmentがやっぱり好きという方も沢山いらっしゃることでしょう。

もちろん、ウィンドウマネージャ、統合デスクトップ環境はこれ以外にも数多くあります。昨今の各種Linuxディストリビューションでは、標準でインストールすると、再起動した後は何もしなくてもGNOMEやKDEの環境になっているということも少くありません。そして気付かない内に、そのままGNOMEとsawfishの組み合わせで使っている方も多いのではないかと思います。

今やsawfishも、GNOME環境での標準ウィンドウマネージャとなってしまった感があり、あまりメジャーでないころからささやかながらその布教 ^H^H 宣伝活動に関わってきた筆者としては、感慨深いものがあります。ちなみにsawfishの作者John Harperは、今では米Eazel社に入社しており、さらに精力的にsawfishの開発を続けています。

sawfishは、その当初からEnlightenmentのように自由にウィンドウフレームをデザインできる機能(=完全なテーマ機能)を持っています。なおかつ、ウィンドウマネージャにとって重要な、基本的機能のみを実装するミニマリズムを基本方針として発展してきたという特徴があります。「rep」というLisp言語の一種でウィンドウマネージャの機能の大部分が書かれているので、ユーザーがrepコードを書いて拡張することが比較的容易な点も特徴の1つです。しかも、GNOMEと非常に相性が良い上に、リソースの消費が比較的少ないという長所を持ちます。また、開発者John Harperの開発意欲も非常に高く、開発速度が速いので、あれよあれよという間に世界中にユーザーを増やしてきました。

sawfishについては、今やさまざまなリソース([2])があり、ささやかながら筆者のWebサイト「Sawfish jp」([3])もありますので、その詳細についてはここでは割愛します。

*1 もちろん皮肉ではありません。筆者もマシンによってはそのような環境を用意しています。ただ筆者はそのような場合、ウィンドウマネージャにはblackboxを使っています。

2 まずは簡単な模様替えから

最初は、「既製品」を利用することから説明しましょう。最近では、どのウィンドウマネージャでもthems org.にテーマが集められており、sawfishも同じ状況にあります。とりあえずs.t.o.に行けば、既に多くのさまざまなテーマが集められているので、適当に好きなものを選択してダウンロードすればいいでしょう。

そして、ダウンロードしたテーマアーカイブを適切なディレクトリに置くのですが、sawfishではシステム全体で使うテーマと、各ユーザーごとに使うテーマでは置き場所が違います。システム全体では、system-theme-directoryを評価すると得られるパスに置きます。インストール時に特に指定していなければ、通常は

```
/usr/share/sawfish/{version}/themes/
```

となっていると思います。各ユーザーごとの場合は、user-theme-direcotryを評価すると得られるパスです。特に指定していなければ、

```
~/.sawfish/themes/
```

となっています。この変数の値を別の値やディレクトリに定義することもできますが、何か特別な理由があるのでなければ、ほとんどの場合変える必要はないでしょう。

ここで「評価する」という用語を使いましたが、実際には「sawfish-client（sawfishシェル）」というものを使っています（実行例1）。sawfish-clientを使うと、repコードを実行（評価）することができます。このrepインタプリタ内では、Tabキーによる補完機能や簡単なヘルプ機能（,helpと試しに入力してみてください）もあり、テーマを作るときにもいろいろと役立ちます。

さて、肝心のテーマの置き場所ですが、特に理由なければ各ユーザーごとにテーマを管理するのをお勧めします。もし~/.sawfish/themes/ディレクトリがなければ、下のようにして作っておいてください。

```
$ mkdir -p ~/.sawfish/themes
```

そして、s.t.o.からダウンロードしたアーカイブをその中に置きます。テーマを改造したい、中身を見たいということでなければ、そのアーカイブのsuffixが、「.tar.bz2」や「.tar.gz」、あるいは「.tar.Z」だろうが、展開を行う必要はまったくありません。

テーマを適切なディレクトリに置いたら、ウィンドウメニューの［フレームスタイル］から、あるいはsawfish-uiの［外観（Appearance）］からそのテーマを選択します。sawfish-uiでは、ルートウィンドウメニューから［カスタマイズ］→［全体］と選択しても可能です。またGNOMEを使っているなら、GNOMEコントロールセンターからsawfishのセクションを選択すると実行できます。

ウィンドウメニューの場合は、選択と同時にそのウィンドウで変更されます。sawfish-uiの場合は、任意のテーマを選択した後に「適用」ボタンをクリックすることで、ウィンドウ全部がそのテーマになります。

3 オリジナルテーマを作る！

「既製品ではもう我慢できない」、「僕の（私の）センスに合うテーマが見つからない」。ここからは、そんな方のために、オリジナルのテーマを作る方法を紹介します。

●テーマの中はどうなっているの？

具体的にテーマの製作方法に入る前に、まず、sawfishでのテーマの中身がどうなっているか見てみましょう（実行例2）。

実行例1　sawfish-client（sawfishシェル）

```
$ sawfish-client -
sawfish 0.34, Copyright (C) 1999-2000 John Harper
sawfish comes with ABSOLUTELY NO WARRANTY; for details see the file COPYING

Enter ‘,help’ to list commands.
user> user-theme-directory
"~/.sawfish/themes"
user> system-theme-directory
"/usr/share/sawfish/0.34/themes"
user> ^D
```

（筆者注：^Dは、実際にはCtrlキー＋d）

実行例2　sawfishテーマの中身の例

```
$ pushed /usr/share/sawfish/0.34/themes/
...
$ ls
absolute-e  brushed-metal  gradient  gtk  microGUI  mono  simple  smaker
bash$ ls absolute-e/
README     README.ja              bar_hilited_active.png  theme.jl
README.gl  README.pl              bar_normal.png
README.it  bar_clicked_active.png  bar_normal_active.png
```

実行例3　使える画像ファイル形式の確認

```
$ ldd `which sawfish`
    librep.so.9 => /usr/lib/librep.so.9 (0x40018000)
    libdl.so.2 => /lib/i686/libdl.so.2 (0x40074000)
    libgmp.so.3 => /usr/lib/libgmp.so.3 (0x40077000)
    libm.so.6 => /lib/i686/libm.so.6 (0x40099000)
    libImlib.so.1 => /usr/lib/libImlib.so.1 (0x400b8000)
    libjpeg.so.62 => /usr/lib/libjpeg.so.62 (0x400f1000)
    libtiff.so.3 => /usr/lib/libtiff.so.3 (0x40111000)
    libungif.so.4 => /usr/lib/libungif.so.4 (0x40154000)
    libpng.so.2 => /usr/lib/libpng.so.2 (0x4015c000)
    libz.so.1 => /usr/lib/libz.so.1 (0x40187000)
    libXext.so.6 => /usr/X11R6/lib/libXext.so.6 (0x40196000)
    libSM.so.6 => /usr/X11R6/lib/libSM.so.6 (0x401a4000)
    libICE.so.6 => /usr/X11R6/lib/libICE.so.6 (0x401ad000)
    libX11.so.6 => /usr/X11R6/lib/libX11.so.6 (0x401c3000)
    libnsl.so.1 => /lib/i686/libnsl.so.1 (0x4029d000)
    libc.so.6 => /lib/i686/libc.so.6 (0x402b5000)
    /lib/ld-linux.so.2 => /lib/ld-linux.so.2 (0x40000000)
```

実行例2のようにテーマディレクトリの中にはREADME*、theme.jl、画像ファイルが置いてあるのが確認できます。

・README*

テーマの簡単な説明が各言語で書かれています。LANGが適切なものになっていて、その言語のREADME.{LANGの最初の部分、例えばja}ファイルがあれば、その内容はsawfish-uiのテーマ選択の説明のところで表示されます。

テーマを作るときは「README」ファイルを必ず用意して、その中身は英語で書きます。日本語の説明も用意したければ、「README.ja」ファイルを用意します。READMEには、日本語で説明を書いてはいけないので注意しましょう。またREADMEの中身は、ごく短かいもので結構です。例えばテーマの名前、製作日時、対応するsawfishのバージョン、作者の名前やメールアドレス、WebページのURLアドレスなどを書きます。

・画像ファイル

ウィンドウのタイトルバーのボタン、ウィンドウフレーム枠などで利用するアイコンの画像です。画像ファイル形式としてどのようなものが使えるかは、実行例3のようにして確認できます。これは筆者の環境での例ですが、JPEG、TIFF、GIF、PNGなどを利用できるらしいことが分かります。おそらくほとんどの環境でこのようになっていることでしょう。ただ、確かにsawfishではさまざまな画像ファイル形式が利用

表1　フレームタイプ

タイプ名	内容
default	タイトルバー、フレーム枠すべてのパーツが揃っています。通常のウィンドウで使います。
transient	フレーム枠だけを持ちます。ダイアログウィンドウなどの、一時的(transient)なウィンドウで使います。
shaped	一部アプリケーション(gqmpeg、gkrellm、xmms)では、ウィンドウの形状が完全な矩形でなく、アプリケーション側でウィンドウの外枠の形状を制御するものがあります。これらのアプリケーションのウィンドウでは枠を付けても意味がないのでタイトルバーだけを持ちます。
shaped-transient	shapedなウィンドウで、特にダイアログなどの一時的なウィンドウである場合には、このタイプとなります。上の部分のフレーム枠(タイトルバーではありません)だけ持ちます。
shaded	MacOSでは、ウィンドウのタイトルバーをダブルクリックするとウィンドウが折り畳められて(shadeされて)、タイトルバーだけの表示になる機能があり、それと同じ役割を果たします。
shaded-transient	一時的なウィンドウの場合の、shadedフレームタイプです。こちらはウィンドウの上の部分のフレーム枠だけが表示されます。

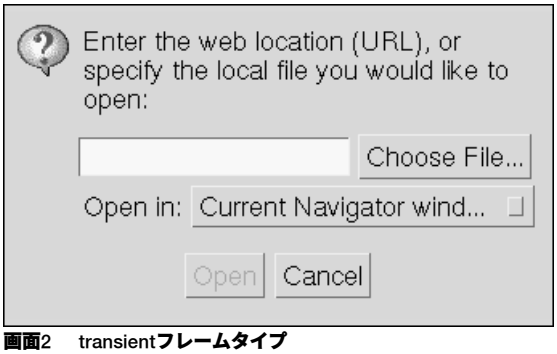
できますが、s.t.o.からダウンロードしたアーカイブを展開して中身を見れば分かるように、ほとんどのテーマでPNG形式が使われています。従って、特別な理由がない限り、PNGを利用することをお勧めします。

・theme.jl

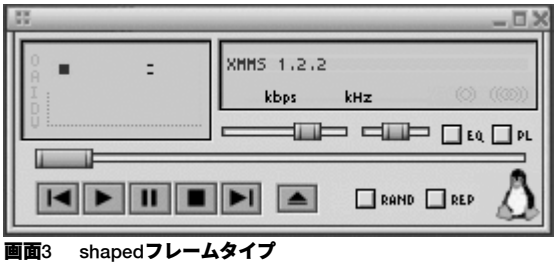
これがテーマの実体であり、一番重要なファイルとなります。このファイルの中でウィンドウフレーム枠、ボタンなどについて、どの画像を使うかを逐一定義していきます。

sawfishではウィンドウの種類、画像について個々のパーツの組み合わせをまとめたものを「スタイル」と呼びます。sawfishでは、ウィンドウごとにこのスタイルを指定することができるようになっています。

画面1　defaultフレームタイプ



画面2　transientフレームタイプ



画面3　shapedフレームタイプ

実は、名前こそtheme（テーマ）となっていますが、theme.jlで定義しているのは、厳密に言えばこの「スタイル」の方です。そして、デスクトップ全体にわたって一貫して特定のスタイルを利用することが、他のウィンドウマネージャでいう「テーマ」と同義のものになっています。

またtheme.jlを見ればわかるように、sawfishでの「テーマ」は、単にウィンドウのフレーム枠のパーツの定義（スタイル）のことを指していて、他のウィンドウマネージャ（例えばWindow Maker）のテーマのような、背景画像や効果音の指定などは「含まれていません」。

●もっと詳しく知りたい！

前述の通り、theme.jlのスタイル定義には、各ウィンドウの状態、種類に応じた定義がそれぞれ必要です。sawfishでは、具体的には以下のようなウィンドウの状態、種類を定義しています。

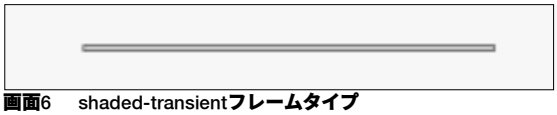
・フレームタイプ

ウィンドウの各状態に対応したフレームの種類のことで、表1と画面1〜画面6のようなものがあります。スタイルを定義するには、まず表1の個々のフレームタイプについて、それぞれ必要なパーツ、アイコン画像などを逐一指定する必要があります。

実際には、フレームタイプの中では「default」タイプだけがすべてのフレームパーツを持ちますので、まずdefaultタイプについて全パーツを定義し、それを他のフレームタイプ

画面4　shaped-transientフレームタイプ

画面5　shadedフレームタイプ



画面6　shaded-transientフレームタイプ

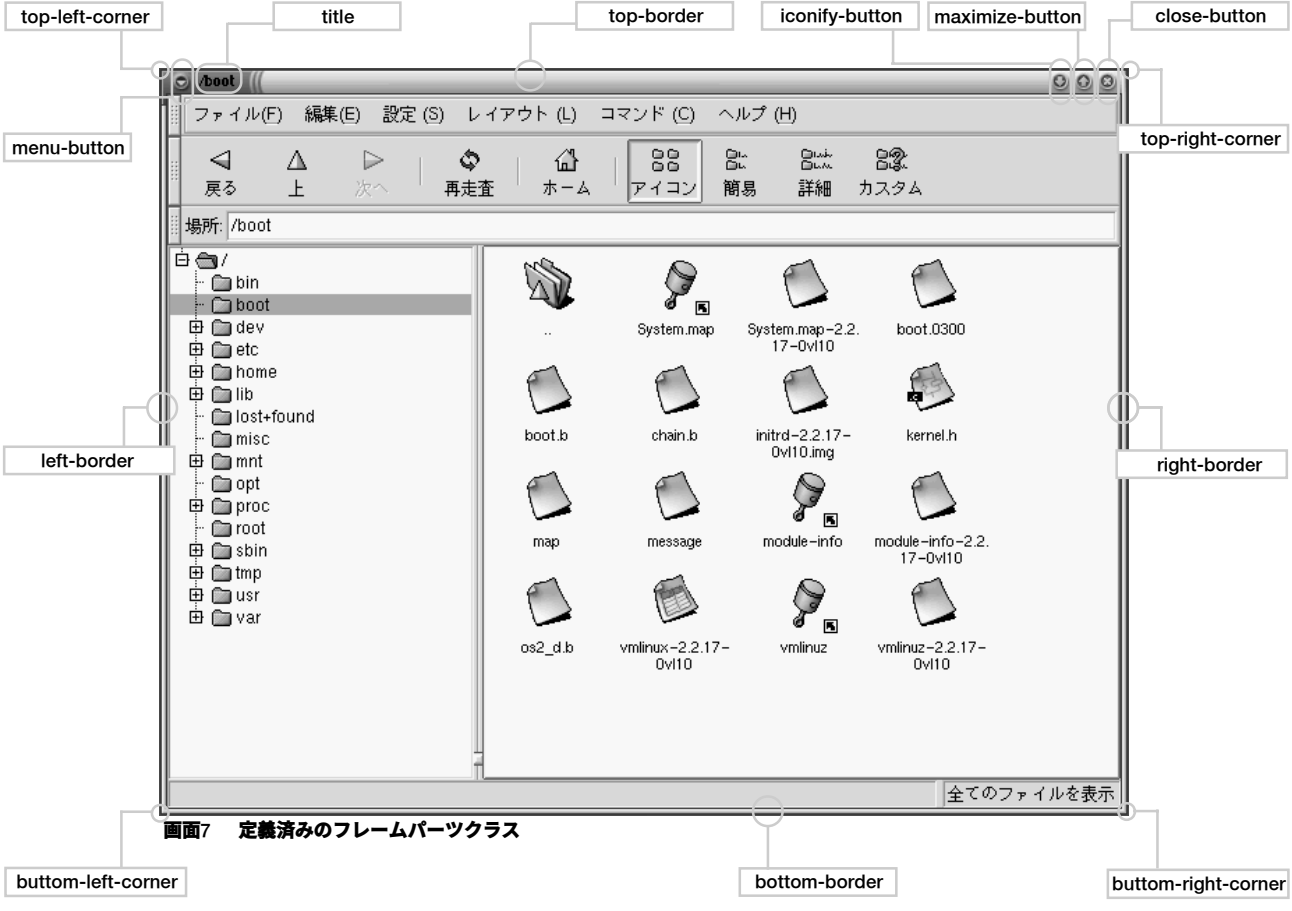
でそのまま流用するほうが楽です*2。

またtheme.jlでは、必ずしもすべてのフレームタイプについて定義する必要はありません。特にshaped*フレームタイプについては、s.t.o.から入手できるテーマでも指定のないものが多数見受けられます。その場合は、未定義のフレームタイプではどのフレームタイプの定義を利用するかという指定を、sawfish-uiの[外観]→[フレームタイプフォールバック]の個所で指定できます。特定のフレームタイプについて指定がない場合、デフォルトでは自動的に別のフレームタイプが使われるようになっていますが、多くのテーマ同様、default、transient、shapedなどについてはtheme.jl内できちんと指定しておくほうが良いようです。

・フレームパーツ

ウィンドウフレーム枠を構成する各部品のことです。ほとんどのテーマでは画像ファイルが使われていますが、rep(Lisp)で独自の描画関数を書いて利用することも可能です。しかし、筆者自身はそのようなテーマを作ったり、そういったテーマをあまり見たことがないので、画像ファイルを利用することを前提に説明していきます。

フレームパーツには何でも指定できますが、一般的なパーツ、例えば最小化や最大化ボタン、閉じるボタン、リサイズ用の角枠などについては、あらかじめ「クラス」といういくつかデフォルトの属性を持つパーツが定義されています。「クラス」を利用すると、クリックやドラッグをしたときにどんなア



画面7　定義済みのフレームパーツクラス

*2　もちろん、単に流用するのではなく、フレームタイプごとに異なる指定もできます。例えばmicroGUIでは、defaultとtransientではタイトルバーの位置が異なります。

クラス名	内容
title	ボタンを除いた、アプリケーションの名前などが表示されるタイトルバーの一部分のこと
menu-button	ウィンドウメニューをポップアップするためのボタン
close-button	ウィンドウを閉じるためのボタン
iconify-button	ウィンドウを最小(アイコン)化するためのボタン
maximize-button	ウィンドウを最大化するためのボタン
top-border	フレーム枠の上の部分のリサイズのための細い部分
left-border	フレーム枠の左側部分のリサイズのための細い部分
right-border	フレーム枠の右側部分のリサイズのための細い部分
bottom-border	フレーム枠の下側部分のリサイズのための細い部分
top-left-corner	フレーム枠の左上部分のリサイズのための小さな部分
top-right-corner	フレーム枠の右上部分のリサイズのための小さな部分
buttom-left-corner	フレーム枠の左下部分のリサイズのための小さな部分
buttom-right-corner	フレーム枠の右下部分のリサイズのための小さな部分

表2
定義済みの
フレームパーツクラス

クションを起すかということを独自にコーディングする必要がなくなりますので、素直にこれを使うことにします。

定義済みのフレームパーツクラスには、表2と画面7のようなものがあります。各パーツの動作はマウス左ボタンクリック時のものです。なお、これらのフレームタイプやデフォルトのフレームパーツクラスなどについては、

```
/usr/share/sawfish/{version}/lisp/sawfish/wm/frames.jl
```

で定義されていますので、興味のある方はそちらを参照してみてください。

4 テーマ作成の手順

それでは、実際にテーマを作る手順について説明していきましょう。今回は、KDE2のウィンドウマネージャのデフォルトの外観をsawfishテーマで模倣することにします。

まず以下のようにテーマのディレクトリを用意し、README(およびREADME.ja)を用意します。

```
$ cd ~/.sawfish/themes/
$ mkdir kde2
$ cd kde2/
$ vi README
```

READMEの内容は、実行例4のように書きました。

実行例4

```
This is a sawfish theme named kde2
ported from the same name theme of icewm.


Created by
SATO Satoru - ssato@redhat.com, ss@gnome.gr.jp
              http://sawfish.gnome.gr.jp
```

●アイコンの画像ファイルを準備

次に、各フレームパーツに使うための画像ファイルを用意します。Gimpなどを駆使して独自に作るなり、他のウィンドウマネージャや、sawfishの他のテーマのものを流用するなりします。

KDE2のテーマなので本物のKDE2から借用したいところですが、残念ながら、ソースを取ってきて画像ファイルを探しても、どこにあるのかよく分かりませんでした。そこで、ちょうど「Icewm Themes Org」([4])で見付けた、Icewm用のKDE2テーマから画像ファイルを流用することにしました。

Icewmでのボタン画像は、クリックと、通常時の画像が組になってXPM形式で用意されています。一方sawfishでは、個々の状態、つまり属性に対して1つの画像ファイルを利用するのが普通です。従って、このIcewmのKDE2テーマの画像を利用するには、多少加工する必要があります。

なお、実際にフレームを描画する際には、フレームパーツクラスのうち「title」や「*-border」に使う画像は必要な大きさに拡大されますが、「*-button」はそのままの大きさで使われます。そのため、画像を作るときに大きさを揃える必要があることに注意してください。

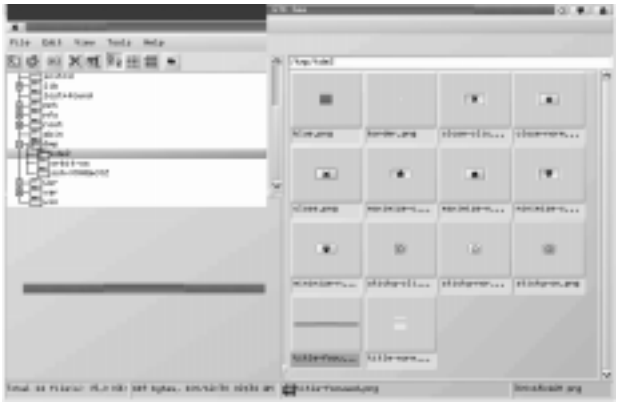
画像作成、あるいは既存の画像を加工するときには、Gimpのレイヤ機能を使うと、ボタンすべてについて一貫した処理が行えて便利です(画面8)。特に、ボタンを作るときは本当に便利、というより、筆者の場合はこれなしでは作成できません。テーマのための画像を作るときには、ぜひGimpを使うことをお勧めします。

そして、加工して最終的にできた画像ファイルが、画面9になります。

●パーツの設定

必要な画像を用意できたところで、theme.jlを書き始めることにしましょう。theme.jlには、rep(Lisp)のコードを書き

画面8　GIMPでレイヤーを使ってボタン画像を加工



画面9　加工後のボタンなどの画像ファイル

ます。「Lispってナニ？」という方もいるはずなので、身近な例を挙げておくと、Emacsの設定ファイルにあたる「.emacs」の中の言語が、「Emacs Lisp」コードとなっています。

また、Lispという言語があるのは知ってはいても、あまり慣れていないという方も多いでしょう。そのような方は、sawfishのプログラミングマニュアルや、librepのinfoマニュアル、Emacs Lispのチュートリアルなどを参考にしてください。本稿では、特にLispの構文については説明しません。なじみのない用語も出てくるかもしれませんが、これについてはご了承願います。

プログラミングマニュアルについては、sawfish jpのdocumentationコーナーからたどれる、筆者の稚拙な :(日本語訳があります。ただ残念ながら、必ずしもオリジナルの英語原文の更新に追従できているわけではありません。従って、日本語マニュアル内での不明な点などについては、必ず原文も参考にしてください。

なお、theme.jlを書くだけなら、ほとんどパターンの定まったコードを書くだけですみます。Lispについて知っていなくても、他のテーマのtheme.jlを真似たりすれば、それほど難しい作業とはなりません。筆者も、初めてsawfishのテーマを作ってs.t.o.に投稿したときは、ほとんどLispについて何も知りませんでした。

それでは、最初に下準備からいきましょう。sawfish内部のrepインタプリタの中でイメージオブジェクトとして扱うために、各パーツの画像ファイルを読み込む必要があります。これには以下のように関数make-imageを使います。

```
(make-image "border.png")
```

ほとんどのテーマでは、theme.jlの冒頭部分で、まずtop-images、menu-images、close-imagsなどといった独自に定義した変数に、各ウィンドウフレームのパーツごとに使うイメージオブジェクトを割り当てています。これは、その後でフレームタイプを定義するときに、長々と画像ファイルを読み込むコードを書く必要がないようにするためです。全体のコードもスッキリする上に便利なので、今回のKDE2テーマでも同じようにすることにします。

・フレーム枠(*-border)に使う画像

[top, bottom, left, right]-imagesは、それぞれウィンドウの上部、下部、左、右のフレーム枠に利用する画像ファイルを参照するために独自に定義した変数です。KDE2では、作成を簡単にするために2×2のただ背景を白くしただけの画像ファイルを利用することにしました。それがリスト1の11行目〜16行目の部分です。このリストは、kde2のテーマ(kde2.tar.gz)の一部として付録CD-ROMに収録してありますので、自作の際にはそちらも参照してください。

枠に使う画像で注意すべき点は、実際にウィンドウフレームが描画されるときには、画像の縦横比が保たれないことです。つまり、画像は適当な大きさに拡大されて利用されることになります。

[top-left, bottom-left, top-right,bottom-right]-imagesは、それぞれ、上部左、下部左、上部右、下部右のフレーム角枠に使う画像ファイルを参照するために定義した変数です。これについても、簡単にするために、top-imagesなどで利用している画像と同じものを使うことにします(リスト1の18行目〜22行目)。

・ボタン(*-button)に使う画像

ボタンやタイトルバー(*-button)には、通常(normal)、通常およびフォーカス時(normal,active)、強調表示およびフォーカス時(hilited,active)、クリック時(clicked,active)などの各状態に応じた画像をそれぞれこの順で対応させ、独自に定義した変数[menu,iconify,maximize,close]-

リスト1　完成したtheme.jl

```
001:;; kde2/theme.jl (for sawfish v0.34)
002:;; $Id: $
003:
004:;; Copyright (C) 2000 SATO Satoru <ssato@redhat.com>
005:;; Licence: GPL v2 or later
006:
007:
008:(let*
009: (
000:
011:;; ------------------------- frame parts images ------------------------
012:        ;; [top,bottom, left, right] border, 2x2
013:     (top-images    (make-image "border.png"))
014:     (bottom-images (make-image "border.png"))
015:     (left-images   (make-image "border.png"))
016:     (right-images  (make-image "border.png"))
017:
018:     ;; [top-left,bottom-left, top-right, bottom-right] border, 2x2
019:     (top-left-images     (make-image "border.png"))
020:     (bottom-left-images  (make-image "border.png"))
021:     (top-right-images    (make-image "border.png"))
022:     (bottom-right-images (make-image "border.png"))
023:
024:     ;; example:
025:     ;; (menu-imags (list (make-image "normal") (make-image "normal_active")
026:     ;;           (make-image "hilited_active") (make-image "clicked_active")))
027:     ;;
028:     ;; (title-images (list (set-image-border (make-image "t2-.png") 10 10 0 0)
029:     ;;                     (set-image-border (make-image "t2.png") 10 10 0 0)))
030:
031:     ;; 15x15, 300x15
032:     (title-images (list
033:            (make-image "title-normal.png")
034:            (set-image-border (make-image "title-focused.png") 100 100 0 0)))
035:
036:     ;; 28x15
037:     (close-images (list (make-image "close-normal.png")
038:                         (make-image "close-normal.png")
039:                         (make-image "close-normal.png")
040:                         (make-image "close-clicked.png")))
041:
042:     ;; 28x15
043:     (maximize-images (list (make-image "maximize-normal.png")
044:                            (make-image "maximize-normal.png")
045:                            (make-image "maximize-normal.png")
046:                            (make-image "maximize-clicked.png")))
047:
048:     ;; 28x15
049:     (iconify-images (list
050:            (make-image "minimize-normal.png") (make-image "minimize-normal.png")
051:            (make-image "minimize-normal.png") (make-image "minimize-clicked.png")))
052:
053:     ;; 18x15, these images is used for menu-button temporarily
054:     (menu-images (list
055:            (make-image "sticky-normal.png") (make-image "sticky-normal.png")
056:            (make-image "sticky-normal.png") (make-image "sticky-clicked.png")))
057:
058:
059:;; ------------------------ frame definitions ------------------------
060:
061:  (frame `(
062:     ((background . ,close-images)
063:      (top-edge . -15) (left-edge . 0)
```

リスト1　続き

```
064:       (class . close-button))
065:      ((background . ,title-images)
066:       (left-edge . 28) (right-edge . 74) (top-edge . -15)
067:       (foreground . "white") (text . ,window-name)
068:       (x-justify . center) (y-justify . center)
069:       (class . title))
070:      ((background . ,menu-images)
071:       (right-edge . 56) (top-edge . -15)
072:       (class . menu-button))
073:      ((background . ,iconify-images)
074:       (right-edge . 28) (top-edge . -15)
075:       (class . iconify-button))
076:      ((background . ,maximize-images)
077:       (right-edge . 0) (top-edge . -15)
078:       (class . maximize-button))
079:      ((background . ,left-images)
080:       (left-edge . -2) (top-edge . -15) (bottom-edge . 0)
081:       (class . left-border))
082:      ((background . ,right-images)
083:       (right-edge . -2) (top-edge . -15) (bottom-edge . 0)
084:       (class . right-border))
085:      ;((background . ,top-images)
086:      ; (left-edge . 0) (right-edge . 0) (top-edge . -2)
087:      ; (class . top-border))
088:      ((background . ,bottom-images)
089:       (left-edge . 0) (right-edge . 0) (bottom-edge . -2)
090:       (class . bottom-border))
091:      ((background . ,top-left-images)
092:       (left-edge . -2) (top-edge . 0)
093:       (class . top-left-corner))
094:      ((background . ,top-right-images)
095:       (right-edge . -2) (top-edge . 0)
096:       (class . top-right-corner))
097:      ((background . ,bottom-left-images)
098:       (left-edge . -2) (bottom-edge . -2)
099:       (class . bottom-left-corner))
100:      ((background . ,bottom-right-images)
101:       (right-edge . -2) (bottom-edge . -2)
102:       (class . bottom-right-corner))))
103:
104:  (shaped-frame `(
105:      ((background . ,close-images)
106:       (top-edge . -15) (left-edge . 0)
107:       (class . close-button))
108:      ((background . ,title-images)
109:       (left-edge . 28) (right-edge . 74) (top-edge . -15)
100:       (foreground . "black") (text . ,window-name)
111:       (x-justify . center) (y-justify . center)
112:       (class . title))
113:      ((background . ,menu-images)
114:       (right-edge . 56) (top-edge . -15)
115:       (class . menu-button))
116:      ((background . ,iconify-images)
117:       (right-edge . 28) (top-edge . -15)
118:       (class . iconify-button))
119:      ((background . ,maximize-images)
120:       (right-edge . 0) (top-edge . -15)
121:       (class . maximize-button))))
122:
123:  (transient-frame `(
124:      ((background . ,left-images)
125:       (left-edge . -2) (top-edge . -2) (bottom-edge . 0)
126:       (class . left-border))
```

リスト1　続き

```
127:      ((background . ,right-images)
128:       (right-edge . -2) (top-edge . -2) (bottom-edge . 0)
129:       (class . right-border))
130:      ((background . ,top-images)
131:       (left-edge . 0) (right-edge . 0) (top-edge . -2)
132:       (class . top-border))
133:      ((background . ,bottom-images)
134:       (left-edge . 0) (right-edge . 0) (bottom-edge . -2)
135:       (class . bottom-border))
136:      ((background . ,top-left-images)
137:       (left-edge . -2) (top-edge . 0)
138:       (class . top-left-corner))
139:      ((background . ,top-right-images)
140:       (right-edge . -2) (top-edge . 0)
141:       (class . top-right-corner))
142:      ((background . ,bottom-left-images)
143:       (left-edge . -2) (bottom-edge . -2)
144:       (class . bottom-left-corner))
145:      ((background . ,bottom-right-images)
146:       (right-edge . -2) (bottom-edge . -2)
147:       (class . bottom-right-corner)))))
148:
149:
150:  (add-frame-style 'kde2
151:      (lambda (w type)
152:        (case type
153:             ((default) frame)
154:             ((transient) transient-frame)
155:             ((shaped) shaped-frame)
156:             ((shaped-transient) transient-frame))))
157:)
```

imagesで、まとめて指定します(リスト1の36行目～58行目)。

また「本物のKDE2」には、タイトルバー部分にstickyボタンというものがあります。sawfishでも同様の機能があり、ウィンドウメニューの[切替]→[居座る]を選択すると、ワークスペースを移動しても、そのウィンドウは同じ場所に「居座って」移動しなくなります。

しかしsawfishでは、標準ではsticky-buttonクラスは用意されていませんので、実装するには自分で定義しなければなりません。多少複雑なコードを書かなくてはならないので、stickyボタンとしてではなく、代わりにmenuボタンとして定義することにします。

・タイトルバー

タイトルバー(tittle)では、2つの状態を考え、それらをまとめて、独自に定義した変数title-imagesに指定することにします。

タイトルバーを描画するときは元の画像は拡大されますが、元の画像の一部の縦横比を保存したい場合は、画像の大きさを指定します。これは、以下のようにset-image-border関数を使います。

```
Function: set-image-border image left right top bottom
```

set-image-border関数に指定するのは、画像の端の一部の、拡大されては困る、縦横比を保ちたい部分の幅、高さなどを定義します。今回フォーカスされたウィンドウのタイトルバーに使う画像は、両端からそれぞれ100だけの幅の部分が横方向に拡大されては困るので、そのように指定します(リスト1の31行目～35行目)。

●フレームを定義する

画像の準備を整えたら、実際のウィンドウフレームを想像しながら、各パーツの属性(配置など)を指定していきます。

まず、defaultフレームタイプについて、先ほど用意した*-images変数を使って、各パーツを割り当てていきます(リスト1の59行目～103行目)。

パーツの割り当ては、順にリストを作っていくだけです。各リストには、背景(画像あるいは色などの指定)、配置する位置、クラスを指定します。配置する位置については、sawfishによるフレーム枠がない、本来のウィンドウ枠を基本に考えます。つまり、ウィンドウマネージャが起動していな

表3　titleでのパーツの配置例

位置	書式例	内容
title左端	(left-edge . 28)	titleパーツの左側にはcloseボタンを配置する予定なので、その幅の分(28)だけ内側に指定します。
title右端	(right-edge . 74)	titleパーツの右側には、menu(sticky)、iconify、maximizeボタンを配置する予定なので、その幅の分(18+28x2)だけ内側に指定します。
title上端	(top-edge . -15)	titleパーツの上端は、ウィンドウ内枠よりもこの長さ分だけ上に指定します。この数字を正にすると、ウィンドウの内容のうち、上端の一部を画像で覆うことになります。

いときに、Xがアプリケーションに割り当てるウィンドウの内枠の枠線位置を0とし、それよりも外側に向うときに「-」(マイナス)、内側に向うときに「+」(プラス)となります。そして、その数字を*-edgeと組み合せたコンスセルとして定義することで、そのパーツの位置が定まります。

コンスセルというのは、Lispの特殊なオブジェクトのことです。テーマ作成では、「(変数 . 値)」のように「()」の中を「.」(ピリオド)で区切って、2つの要素を並べていることだけ分かればさしつかえありません。例えばtitleでは、パーツの配置は表3のようになります。

この他、配置以外にもさまざまな属性を指定しますが、フレームスタイルを構築する際には、配置ほど重要な属性はありませんので、説明はここでは割愛します。詳しくは、sawfishのプログラミングマニュアルなどを参照してください。

default フレームを定義したら、shaped*、transient*などについても同様に定義していきます。これらは、デフォルトフレームから適切なパーツを削っていけばいいだけなので、それほど難しい作業ではないでしょう。

すべてのフレームタイプに対応するフレーム枠の属性を定義したら、それを実際のウィンドウのフレームタイプに指定します(リスト1の150行目〜156行目)。ここではテーマの名前も指定しますが、そのテーマのディレクトリ名と同じ名前にする必要がある点に注意してください。

画面10　完成したSawfish用テーマ「KDE2」

●テーマを試す

theme.jlを書き終わったら、今度は実際にそのテーマを試してみます。theme.jlのバグはよくあることなので、まずは適当なウィンドウについて、ウィンドウメニューから[フレームスタイル]→[kde2]を選択します。自分で作ったテーマを試したいときには、ここで「kde2」ではなく自作のテーマを選ぶだけでOKです。

theme.jl内に記述間違いがある、あるいは存在しない画像ファイルを参照しているといった場合、ビープ音が鳴り、sawfishのデフォルトテーマ(microGUIなど)か、前に指定したテーマにフォールバックします。この場合はtheme.jlをよく見直し、画像を確認して再度試します。

正しく動作し、そのテーマのフレームが表示された場合でも、何か気に入らない部分があるようなら、再度画像をチェック、試すということを完成するまで繰り返します。そうやって完成させたのが、画面10です。

なおsawfishは、1度テーマを読み込むとそれを記憶しています。再試行する際には、テーマを再読み込みして、変更がすぐ反映されるようにする必要があります。再読み込みするには、sawfish-client上で以下のように打ち込みます。

```
$ sawfish-client -
       ⋮
user > (reload-frame-style 'kde2)
user > ^d
```

もちろん、ここで「kde2」となっている個所には現在作っているテーマの名前を入力します。また「^d」は、実際にはCtrlキー+dを押しています。またテーマの中には、ウィンドウメニューを表示するためのボタンがないものがありますが、その場合は「Altキー+ウィンドウ内部をマウス左クリック」で、ウィンドウメニューを表示できます。

5 sawfish-themerとは？

ここまでで、theme.jlを書くことで、オリジナルのテーマを作れることが分かったと思います。でも、何やらいろいろと大変そうだという印象を与えてしまったかもしれません。現状では、Enlightenmentなど他のウィンドウマネージャでも、設定ファイルを自分で書いてsawfishのようなテーマを作るということは、それほど珍しいことではないようです。

しかし実は、sawfishの場合はtheme.jlを手で書かなくてもテーマを作れるアプリケーション、「sawfish-themer」がsawfish本体と一緒に配布されています（画面11）。

sawfish-themerはGUIベースの設定画面を採用しており、そのおかげでtheme.jlを自分で書くときよりも比較的簡単にテーマを作れます。メニューや選択ダイアログからさまざまな項目を選択し、必要な内容をテキストフィールドに入力するといった作業を行うことで、theme.jlを設定できるようになっています。今回は、残念ながらsawfish-themerについて詳細な説明はできませんが、その使用手順について、ごく簡単に紹介しておきます。基本的には、theme.jlを書く場合と同じです。

① 下準備

theme.jlを自分で書く方法と同様に、画像やテーマ名のディレクトリを用意しておきます。

② sawfish-themerを起動

用意したテーマ名のディレクトリに移動し、sawfish-themerを起動します。

③ [詳細]タブ

テーマ名や、そのテーマについて簡単な説明を書きます。この内容がそのままREADMEファイルに書き出されます。

④ [パターン]タブ

各パーツごとに、さまざまな状態に対応するパターンの組み合わせを編集します。これは、theme.jlを書くときに、画像ファイルの組み合わせの変数を用意したのに似ています。

⑤ [フレーム]タブ

フレームタイプごとに、フレームパーツの配置などを編集していきます。

⑥ [割り当て]タブ

作ったフレームを、実際のフレームタイプに割り当てていきます。

⑦ 試行

sawfish-themerのメニューから[試行]を選択して、作ったテーマがどのように表示されるかを試します。なお、このとき表示されるのはdefaultフレームタイプです。

画面11　sawfish-themer

R E S O U R C E

[1] sawfish themes org
http://sawmill.themes.org/

[2] sawfish: an extensible window manager
http://sawmill.sourceforge.net/

[3] Sawfish JP
http://sawfish.gnome.gr.jp/

[4] bb themes org
http://bb.themes.org/

[5] Icewm Themes Org
http://icewm.themes.org/