

FbTk Tutorial

Henrik Kinnunen

9th May 2003

Contents

1	Introduction	3
1.1	History	3
2	Getting started	4
3	Window	7
4	Pixmap	8
5	Resource	9
6	Texture	10
7	Color	11
8	Theme	12
9	Button	14
10	Menu	15
11	Examples	16

Chapter 1

Introduction

1.1 History

FbTk started of with Flubox and is now an important part of it.

Chapter 2

Getting started

The first thing you must do before you do anything with FbTk is to initiate an display connection. This is done by creating one instance of FbTk::App. Example:

```
FbTk::App app;
```

This will initiate a display connection on the default display which is where the DISPLAY enviroment variable points to. To get display connection to another display you just supply the display string. Example:

```
FbTk::App app(":1");
```

If no display connection could be established the FbTk::App::display() will return 0. You can only create one instance of FbTk::App, if you try to create more than one instance will throw std::string.

After a display connection been established you can enter the main event loop: Example:

```
app.eventLoop();
```

This will loop until you call FbTk::App::end().

To catch events you need to have an instance of FbTk::EventHandler: Example (not all EventHandler function visible):

```
class MyEventHandler: public FbTk::EventHandler {
    void handleEvent(XEvent &event);
    void exposeEvent(ExposeEvent &event);
};
```

And to register this to a specific window you call FbTk::EventManager Example:

```
FbTk::EventManager::instance()->add(your_eventhandler_instance, your_window);
```

Don't forget to unregister it when the window dies.

To create a simple window you'll use FbTk::FbWindow. Example: FbTk::FbWindow window(0, 10, 20, 300, 400, ExposeEvent); This will create a window on screen 0, position x = 10 y = 20, with size 300x400 and with event mask ExposeEvent. To view all possible event masks you need to read the Xlib manual.

Simple application (file

```

simple_app.cc
):
// Compile this with: g++ -Wall -g -O2 -IFbTk FbTk/libFbTk.a -lX11 -L/usr/X11R6/lib simple_app.cc
// FbTk headers
#include "FbTk/App.hh"
#include "FbTk/FbWindow.hh"
#include "FbTk/EventHandler.hh"
#include "FbTk/EventManager.hh"

// system headers
#include <iostream>
#include <cstdlib>

// we use names in FbTk area
using namespace FbTk;

// handles events
class MyEventHandler: public EventHandler {

    void exposeEvent(ExposeEvent &event) {
        cerr<<"Got ExposeEvent!"<<endl;
    }

    void buttonPressEvent(ButtonEvent &event) {
        cerr<<"Got ButtonPressEvent!"<<endl;
    }

};

int main(int argc, char **argv) {
    // initiate display connection
    App app;
    if (app.display() == 0) {
        cerr<<"Can't connect to display"<<endl;
        exit(0);
    }

    // create a window at pos 10 10 and with size 100 100
    // and with eventmask ExposeMask ButtonPressMask
    FbWindow window(0, 10, 10, 100, 100, ExposeMask | ButtonPressMask);

    // register eventhandler with window to eventmanager
    MyEventHandler ev_handler;
    EventManager::instance()->add(ev_handler, window);

    // make window visible
    window.show();

    // start main loop

```

```
app.eventLoop();  
  
EventManager::instance()->remove(window);  
}
```

This will initiate display connection and create a window at position 10 10 and with size 100x100 with event mask ExposeMask ButtonPressMask.

Chapter 3

Window

To create a simple window with `FbTk` use `FbTk::FbWindow`.

Chapter 4

Pixmap

To create a pixmap use `FbTk::FbPixmap` Example:

```
FbTk::FbPixmap pm(a_drawable, 20, 20, 16);
```

This will create a pixmap with

`a_drawable`

as reference, size 20x20 and with depth 16 bits per pixel. `FbPixmap` is a `FbDrawable` so you can use normal `FbDrawable` operations on it.

Chapter 5

Resource

To handle resources within your program use `FbTk::Resource` and `FbTk::ResourceManager`. To create a resource item use:

```
FbTk::Resource(FbTk::ResourceManager, defaultvalue, resource_string, alternative_resource_string)
```

Then you need to implement

```
FbTk::Resource<T>::setFromString(const char *str) and FbTk::Resource<T>::getString().
```

Example:

```
FbTk::ResourceManager res_manager;  
FbTk::Resource<int> my_int_resource(res_manager, 10, 'myresource', 'MyResource');
```

`my_int_resource`

will register itself in

`res_manager` and when `res_manager`

load the resource it'll search for "myresource" and "MyResource" in the resource file and call

```
FbTk::Resource<T>::setFromString if it finds the resource string else it'll call FbTk::Resource<T>
```

Use `FbTk::ResourceManager::load(filename)` and `save(filename)` to load and save all resources.

Chapter 6

Texture

To apply texture to windows and pixmaps you need to use `FbTk::Texture` and `FbTk::TextureRender`. With `FbTk::TextureRender` you render the `FbTk::Texture` to a X pixmap. You can use `FbTk::ImageControl` to have rendered `FbTk::Texture` cached. Example:

```
FbTk::ImageControl icontrol(screen_num);
```

Chapter 7

Color

Allocate and handle colors with `FbTk::Color`. You can create a color from a string with

```
FbTk::Color('black', screen_num) or from rgb values FbTk::Color(red, green, blue, screen_num)
```

Chapter 8

Theme

The theme engine of FbTk consist of three parts.

- ThemeItem<T>
- Theme
- ThemeManager

ThemeItem<T> is the basic item in each Theme. Theme holds ThemeItem<T> for a specific screen and theme. ThemeManager holds all Theme instances on all screens. To create a ThemeItem<T> you supply type, FbTk::Theme instance and the themeitem string, so you must create a FbTk::Theme first. Example:

```
class MyTheme:public FbTk::Theme {
    MyTheme(int screen_num):FbTk::Theme(screen_num),
        border_width(*this, "borderWidth", "BorderWidth");
    // this comes from FbTk::Theme and must be implemented
    void reconfigTheme() { /* do reconfigure stuff here */ }
private:
    ThemeItem<int> border_width;
};
```

This will use borderWidth or BorderWidth in your theme file to load

border_width

value. You need to implement the specialized functions in FbTk::ThemeItem<T> for your T-type. In the example above you need to add:

```
template <>
void ThemeItem<int>::load() { }

template <>
void ThemeItem<int>::setDefaultValue() { *(*this) = 0; }

template <>
void ThemeItem<int>::setFromString(const char *str) {
    *(*this) = atoi(str);
}
```

The `FbTk::ThemeItem<T>::load` function is called from `ThemeManager` after `FbTk::ThemeItem<T>::setFromString` so you can load additional stuff. More about that in the advanced section. `FbTk::Theme` will register itself in `FbTk::ThemeManager` so you don't have to worry about register/unregister. To load the theme you call `FbTk::ThemeManager::instance()->load(filename)`, once all theme items in `FbTk::Theme` been loaded the thememanager will call `FbTk::Theme::reconfigTheme()` in which you can do some update stuff that needs to be done.

Chapter 9

Button

Chapter 10

Menu

To create a menu you need to first create a `MenuTheme` that holds the style for the menu and a `FbTk::ImageControl` to cache `FbTk::Texture`. Example:

```
FbTk::MenuTheme mytheme(screen_num);  
FbTk::ImageControl myimagecontrol(screen_num);
```

Once a menu theme and image control exist you can create a simple menu:

```
FbTk::Menu mymenu(mytheme, screen_num, myimagecontrol);  
mymenu.show();
```

To add simple items to the menu:

```
mymenu.insert('hello');  
mymenu.insert('world!');  
mymenu.reconfigure();
```

You must call `FbTk::Menu::reconfigure` after you inserted items, so the menu can rerender it's graphic.

Chapter 11

Examples