

The KSplash Handbook

Teemu Ryttilahti, Brian C. Ledbetter, and Ravikiran
Rajagopal



The KSplash Handbook

Contents

1	Introduction	1
2	Using themes	2
2.1	Using the KControl Module	2
3	How to make themes for KSplash	3
3.1	General	3
3.1.1	Identifying your theme	3
3.1.2	Background files	4
3.2	Options for Theme Engines	4
3.2.1	Default Theme	4
3.2.2	Standard Theme	4
3.2.3	Redmond theme	4
3.2.4	MacX Theme	4
3.2.5	MacClassic Theme	4
3.2.6	2k theme	4
4	Using KSplash From Within Your Own Application	10
4.1	Basic Requirements	10
4.2	Starting KSplash	10
4.3	Showing messages	11
5	Writing new KSplash plugins	12
5.1	Basic Requirements	12
5.2	Building the skeleton framework	12
5.3	Declaration of plugin class	13
5.4	Code for the header file	14
5.5	Implementation of the plugin	16
5.6	Compiling the plugin	17

The KSplash Handbook

6	Questions and Answers	19
7	Credits and License	20
A	Installation	21
A.1	Requirements	21
A.2	Compilation and Installation	21
B	Source code	22
B.1	Listing of <code>theme2k.cpp</code>	22
B.2	Listing of <code>rotwidget.h</code>	24
B.3	Listing of <code>rotwidget.cpp</code>	25

List of Tables

3.2	Default Theme Options	5
3.4	Standard Theme Options	6
3.6	Redmond theme options	7
3.8	MacX Theme Options	8
3.10	MacClassic Theme Options	8
3.12	2k theme options	9

Abstract

KSplash is a nice splash screen that shows the progress of an application that is loading.

Chapter 1

Introduction

KSplash is a nice splash screen that shows the progress of an application that is loading. Please report any problems or feature requests to the KDE mailing lists. The principal features of KSplash:

- Themeable
- Uses plugins for complete customizability
- Can be used by any application that uses DCOP

This handbook will show you how to create themes for use with plugins that are already available. If none of the plugins available satisfy your tastes, you can learn how to customize the appearance of KSplash completely by writing a plugin in C++.

Chapter 2

Using themes

To use themes from [KDE-Look](#), extract them to `/.kde/share/apps/ksplash/Themes/` for a single user, or to `$KDEDIR/share/apps/ksplash/Themes/` to make them available to all users of your system.

You can also use the Splash Screen module under Appearance in the KDE control center to do this automatically.

2.1 Using the KControl Module

This module allows you to install, test and remove KSplash themes.

Down the side of the module is a list of currently available KSplash themes. As you select one, a preview will display in the main part of the window. When you have selected the one you wish to use, press OK or Apply. Press Cancel to exit the module without making changes, and Defaults to restore the system default splash screen.

To install new modules, press Add..., and find the theme on your computer. You do not have to unpack theme files, you can safely select the compressed theme file. Installing a theme does not make it the theme in use until you select it in the list and press either OK or Apply.

Although you can see a preview of the splash screen, you may like to see how it looks in real use, for instance to see what the animation looks like. You can test themes by selecting them in the list and clicking the Test button.

You can also remove themes you no longer wish to use, by selecting them and pressing the Remove button. Note that your user account may not have the right to remove themes installed system-wide. It is also recommended you do not uninstall the Default splash screen.

Chapter 3

How to make themes for KSplash

3.1 General

Making your own themes for KSplash is easy. After you have finished your themes you can post them on the [KDE-Look](#) so that others can use it.

3.1.1 Identifying your theme

Let us create a theme called `MyCoolTheme`. For the theme to be recognized by KSplash, it should be stored in a folder called `MyCoolTheme` under `/.kde/-apps/ksplash/Themes/`. It should have a file called `Theme.rc`, containing the settings of the theme. You can specify large numbers of special things to theme, change the plugin engine to use, and so on. You do not have to use all the settings available; usually, the settings have an acceptable default value. The basic syntax for entries in the `Theme.rc` file is `[option] = [value]` You can find the definitions of the various options in the following sections.

Example 3.1 Simple Theme.rc file

```
[KSplash Theme: MyCoolTheme]
Name = MyCoolTheme
Description = A nice theme using XpLike engine
Version = 1.0
Author = Real Name <realmail@mail.com>
## Use the XpLike engine for this theme.
Engine = XpLike
Show Icon = false
Welcome Text = Loading KDE
```

The KSplash Handbook

After specifying the name, the description and the author of the theme, you should first choose a theme engine (also known as a plugin). Then, you can customize various features of the theme engine by assigning key-value pairs as in the example file above.

IMPORTANT

Ensure that the name of the directory that contains the theme files (`/.kde/apps/ksplash/Themes/MyCoolTheme` in this example) and the identifier (`[KSplash Theme: MyCoolTheme]` in this example) of the theme in the `Theme.rc` file are identical. Otherwise, KSplash will not recognize the theme.

3.1.2 Background files

When KSplash starts, it tries to find a background image for your current screen resolution, if the theme engine uses one. The background image file should be named in the following format: `Background-WWWxHHH.png`.

For example, you might use a file called `Background-1024x768`. If the background image for your screen resolution cannot be found, it tries to resize the original `Background.png` or the file specified in `Theme.rc` to suit the current resolution. Resizing on-the-fly will certainly take some time, so you should provide background images for at least the following sizes: 1280x1024, 1024x768 and 800x600.

3.2 Options for Theme Engines

3.2.1 Default Theme

3.2.2 Standard Theme

3.2.3 Redmond theme

3.2.4 MacX Theme

3.2.5 MacClassic Theme

3.2.6 2k theme

The KSplash Handbook

Name	Argument	Explanation
Always Show Progress	[true/false]	Indicates whether loading progress should be shown. Default is true.
Label Foreground	[color]	Determines what color to use for the statusbar text. Default is #FFFFFF (white).
Icons Flashing	[true/false]	Indicates whether icons should 'flash'. Default is true.

Table 3.2: Default Theme Options

The KSplash Handbook

Name	Argument	Explanation
Statusbar Position	[top/bottom]	Toggles the position of the statusbar on the screen. Default is bottom.
Statusbar Visible	[true/false]	Indicates whether the statusbar should be shown. Default is true.
Progress Visible	[true/false]	Indicates whether loading progress should be shown. Default is true.
Statusbar Font	[fontname]	The font used in statusbar. Default is Helvetica.
Statusbar Font Size	[size]	The font size for the statusbar. Default is 16.
Statusbar Font Bold	[true/false]	Indicates whether the statusbar font should be bold. Default is true.
Statusbar Font Italic	[true/false]	Indicates whether the statusbar font should be italic. Default is false.
Statusbar Foreground	[color]	The foreground color of statusbar. Default is white.
Statusbar Background	[color]	The background color of statusbar. Default is black.
Statusbar Icon	[true/false]	Indicates whether the statusbar should have an icon.
Icons Visible	[true/false]	Indicates whether icons should be visible. Default is true.
Icons Jumping	[true/false]	Indicates whether icons should be jumping. Default is true.
Icon Position	[0-3,10-13]	Position where the icons are shown. Default is bottom-left.
Splash Screen	[name]	Changes the splash screen image that is shown.

Table 3.4: Standard Theme Options

The KSplash Handbook

Name	Argument	Explanation
Background Image	[filename]	User defined background image to use.
User Icon	[Iconname]	Name of standard icon to show for user. Default is <i>go</i> .
Welcome Text	[text]	Text shown in splash screen. Default is "Welcome".
Username Text	[text]	Text shown instead of user's real name.
Welcome Text Position	[x,y]	Position on the screen where the Welcome Text is shown.
Username Text Position	[x,y]	Position on the screen where the username is shown.
Action Text Position	[x,y]	Position on the screen where the current action is shown.
Icon Position	[x,y]	Position on the screen where the user icon is shown.
Show Welcome Text	[true/false]	Toggles showing of welcome text. Default is true.
Show Welcome Shadow	[true/false]	Toggles showing of welcome text's shadow. Default is true.
Show Username	[true/false]	Toggles showing of username. Default is true.
Show Action	[true/false]	Toggles showing of action currently being performed. Default is true.
Show Icon	[true/false]	Indicates whether icon should be shown. Default is true
Use KDM User Icon	[true/false]	Show user's login icon. Default is true.

Table 3.6: Redmond theme options

The KSplash Handbook

Name	Argument	Explanation
Icon Size Minimum	[size]	Assign the minimum size for icons. Default is 16.
Icon Size Maximum	[size]	Assign the maximum size for icons. Default is 64.
Optimized Icon Rendering	[true/false]	Optimize icon rendering. Default is true.
Progress Bar Visible	[true/false]	Default is true.
Progress Bar Position	[top/bottom]	Toggles whether statusbar should be in bottom or top. Default is bottom.
Icons Jumping	[true/false]	Indicates whether icons should be jumping. Default is false.

Table 3.8: MacX Theme Options

Name	Argument	Explanation
Icon Position	[0-3,10-13]	Position of the icons on the screen. Default is bottom left.
Icons Jumping	[true/false]	Indicates whether icons should be jumping. Default is false.
Icons Visible	[true/false]	Indicates whether icons should be visible. Default is true.
Splash Screen	[name]	Changes the splash screen image that is shown.

Table 3.10: MacClassic Theme Options

The KSplash Handbook

Name	Argument	Explanation
Title Background Color	[color]	The background color of title. Default is dark blue.
Title Foreground Color	[color]	The foreground color of title. Default is white.
Status Text Color	[color]	The color of status texts. Default is the same as Title Background Color.
Rotator Color 1	[color]	Defines the color of rotator 1. Default is dark blue.
Rotator Color 2	[color]	Defines the color of rotator 2. Default is cyan.
Rotator Speed	[value]	Defines the speed of the rotator. Default is 30.
Window Title	[text]	Specifies the title text of the window.
Logo File	[filename]	Defines the logo used.

Table 3.12: 2k theme options

Chapter 4

Using KSplash From Within Your Own Application

In this chapter, we describe a simple method for using KSplash as the splash screen for your KDE application. If you do not develop applications for KDE, you can skip this chapter.

4.1 Basic Requirements

Your KDE application must be DCOP-aware. DCOP is the KDE technology used to communicate between applications. If you use the standard [KDE application framework](#), this is taken care of automatically. For information about DCOP and related KDE technologies, please visit the [KDE developers' corner](#).

4.2 Starting KSplash

Before your application starts its computation intensive work, or before it starts loading plugins, etc., invoke KSplash as follows:

```
DCOPClient *c = kapp->dcopClient();
QString error;
QString KSplashName;
int pid = 0;
QStringList args;
args << "--theme=MyCoolTheme" << "--managed";
if (kapp->startServiceByDesktopName("ksplash", args, &error,
&KSplashName, &pid))
{
```

The KSplash Handbook

```
KMessageBox::sorry(0, error, "Unable to invoke KSplash");  
// Some error processing here.  
}
```

We will assume that there is only one instance of KSplash running. Other cases are slightly more complex. Please see the DCOP documentation for further details.

4.3 Showing messages

Before you show any messages, you need to set up the number of steps you will show. For example, the KDE startup procedure uses 7 steps.

```
QByteArray data;  
QDataStream arg(data, IO_WriteOnly);  
arg << someNumber;  
if (!(c->send(KSplashName, "KSplashIface", " ←  
    setStartupItemCount(int)",  
data))  
    // Some error processing here.
```

Whenever you want to display a message with or without an icon, use

```
arg << QString("iconName") << QString("programName") <<  
QString("Some description");  
if (!(c->send(KSplashName, "KSplashIface",  
"programStarted(QString,QString,QString)", data))  
{  
    // Some error processing here.  
}
```

Each time you call `programStarted`, the steps completed is incremented. When your program has finished its startup, do the following to make the splash screen go away:

```
if (!(c->send(KSplashName, "KSplashIface", " ←  
    startupComplete()", data))  
{  
    // Some error processing here.  
}
```

That's it! You don't need anything more to take advantage of all that KSplash has to offer you.

Chapter 5

Writing new KSplash plugins

Writing new KSplash plugins is not difficult. In this chapter, we will write a simple plugin that will emulate the splash screen of a well known operating system. This tutorial assumes that you know the basics of C++, and a little bit of KDE/Qt programming.

5.1 Basic Requirements

We will create a plugin called `2k`. The plugin name is used in various places, and is important that you consistently use it so that the plugin is recognized by KSplash. KSplash plugins are actually dynamically loadable libraries with the following naming convention:

The library should be named as `ksplash+lowercasethemename`. For our theme, it will be `ksplash2k`.

It should have a corresponding desktop file which is named as `ksplash+lowercasethemename.desktop`. For our theme, it will be `ksplash2k.desktop`.

Finally, the object that is returned by the library should be a class which is named `Theme+themename`. For our theme, it will be `Theme2k`.

Do not worry about it if you don't understand all of the above. We will consider each of those points in detail later. The other very important detail is that the plugin class should be derived from `ThemeEngine`.

5.2 Building the skeleton framework

We will use the KDE application framework which will take care of building the plugin and will provide us with platform independence without any work on our part. To do that, make sure you have the `kdesdk` package installed. Run the command `kapptemplate` to produce an application named "2k". It will create a toplevel folder which contains generic files such as `AUTHORS`, etc..

The KSplash Handbook

We are most interested in the subfolder called `2k`. Go into that subfolder and delete all the files there. Now we have the skeleton we require.

The next step is to create a `.desktop` file which, when installed, will tell KSplash that our plugin is available. Consistent with the naming conventions laid out in [the preceding section](#), create a file called `ksplash2k.desktop` in that folder. It should contain the following lines:

```
[Desktop Entry]
Encoding=UTF-8
Type=Service
Comment=KSplash Plugin
Name=KSplash2k
ServiceTypes=KSplash/Plugin
X-KDE-Library=ksplash2k
X-KSplash-Default=true
X-KSplash-PluginName=2k
X-KSplash-ObjectName=Theme2k
```

The `Encoding`, `Type`, `Comment` and `ServiceTypes` are the same for all plugins. The plugin name and the library name follow the conventions noted earlier. The entry `X-KSplash-Default` takes a boolean value which determines whether it is shown in the control panel configuration module by default. Except for some very rare cases, it should be `true`.

5.3 Declaration of plugin class

Now that we have the preliminary work done, let us get into the actual fun part - creating a class that will provide the behavior we want. While we are free to make this class do almost anything we want it to do, there are a few restrictions.

1. Plugin classes must inherit the `ThemeEngine` class.
2. Plugin classes must be named according to the rule: `Theme+PluginName`.
3. Plugin classes should provide a `static` function called `names` that returns a list of names by which it can be invoked.
4. If the plugin can be configured in the control center module, it should provide a `ThemeEngineConfig`-based class for the configuration.
5. Plugin classes must override at least one of the virtual functions `slotSetText`, `slotSetPixmap`, `slotUpdateProgress` and `slotUpdateSteps` to make it usable.
6. The constructor should take the form `ThemeEngine(QWidget *parent, const char *name, const QStringList &args)` so that it can be used with `KGenericFactory`.

The last requirement may seem complicated, but, as we will see later, by adding a single line to your source files, you can usually ignore it.

5.4 Code for the header file

Given the constraints, we will now see what the header file `theme2k.h` looks like this:

Example 5.1 Listing for theme2k.h

```

#ifndef __THEME2K_H__
#define __THEME2K_H__

#include <qlabel.h>
#include <qwidget.h>

#include <kdialogbase.h>
#include <kpixmap.h>
#include <ksplash/themeengine.h>

class RotWidget;

class Cfg2k: public ThemeEngineConfig
{
    Q_OBJECT
public:
    Cfg2k( KConfig * );
};

class ObjKsTheme;
class Theme2k: public ThemeEngine
{
    Q_OBJECT
public:
    Theme2k( QWidget *, const char *, const QStringList& );

    inline const QString name()
    {
        return( QString("KSplash2k") );
    }
    inline const KDialogBase *config( KConfig *kc )
    {
        return new Cfg2k( kc );
    }
    static QStringList names()
    {
        QStringList Names;
        Names << "KSplash2k";
        Names << "ks2k";
        Names << "2k";
        Names << "2000";
        return( Names );
    }
};

public slots:
    inline void slotSetText( const QString& s )
    {
        if( mText && mText->text() != s ) mText->setText( s );
    };

private:
    void initUi();
    void readSettings();

    QLabel *mText;
    RotWidget *mRotator;
    QColor mTbGColor, mTFgColor, mRotColor1, mRotColor2, ←
        mStatusColor;
    int mRotSpeed;

```

Let us analyze the listing above. The `Theme2k` class satisfies the naming conventions, and is inherited from `ThemeEngine`. It provides a `Theme2k::names()`, and has a constructor that takes the required parameters: `Theme2k(QWidget *, const char *, const QStringList&);` and also provides a simple `Theme2k::slotSetText()` method. For the moment, do not worry about the `RotWidget` class. It is a small widget that provides some eye candy for the user. Our plugin is very simple and does not display any icons or show a progressbar. If you would like to display icons, override the `slotSetPixmap` function. Similar functions exist for setting the progressbar range (`slotUpdateSteps`) and incrementing (`slotUpdateProgress`) the current step.

5.5 Implementation of the plugin

We will examine only the relevant parts of the implementation. For a listing of the whole implementation, please see the appendix. The first thing we will do is to get the library requirement out of the way:

Example 5.2 Library requirement

```
K_EXPORT_COMPONENT_FACTORY( ksplash2k, KGenericFactory< ↔  
    Theme2k > );
```

The macro `K_EXPORT_COMPONENT_FACTORY` is declared in `kgenericfactory.h`. Onwards to the constructor! Since this is a very simple plugin, the constructor is pretty straightforward.

Example 5.3 Plugin constructor

```
Theme2k::Theme2k( QWidget *parent, const char *name, const ↔  
    QStringList &args  
    )  
    :ThemeEngine( parent, name, args )  
{  
    readSettings();  
    initUi();  
}
```

The method `readSettings()` illustrates the proper way to obtain your theme settings. (You do want people to use your plugins in their themes, don't you?)

Example 5.4 Obtaining theme settings

```

void Theme2k::readSettings()
{
    if( !mTheme )
        return;

    KConfig *cfg = mTheme->themeConfig();
    if( !cfg )
        return;

    cfg->setGroup( QString("KSplash Theme: %1").arg(mTheme->
        theme()) );

    QColor DefaultTbGColor( Qt::darkBlue );
    QColor DefaultTFgColor( Qt::white );

    mTbGColor = cfg->readColorEntry( "Title Background Color",
    &DefaultTbGColor );
    mTFgColor = cfg->readColorEntry( "Title Foreground Color",
    &DefaultTFgColor );
    mStatusColor = cfg->readColorEntry("Status Text Color", &
        mTbGColor );

    QColor DefaultRot1( Qt::darkBlue );
    QColor DefaultRot2( Qt::cyan );
    mRotColor1 = cfg->readColorEntry( "Rotator Color 1", &
        DefaultRot1 );
    mRotColor2 = cfg->readColorEntry( "Rotator Color 2", &
        DefaultRot2 );

    mRotSpeed = cfg->readNumEntry( "Rotator Speed", 30 );
    mWndTitle = cfg->readEntry( "Window Title", i18n("Please
        wait...") );
    mLogoFile = cfg->readEntry( "Logo File", QString::null );
}

```

Since we like our users, we provide sensible defaults for parameters that are not present in the theme file. Note that we should always set our group to "KSplash Theme: themename" to remain compatible with future theme specifications. The `initUI()` method is not very interesting, as it merely builds up the widgets. Please see the appendix for details.

5.6 Compiling the plugin

Since we decided to use the KDE framework for compiling the plugin, we need to create a `Makefile.am`. It should look like this:

Example 5.5 Listing of Makefile.am

```
INCLUDES = $(all_includes)

kde_module_LTLIBRARIES = ksplash2k.la

ksplash2k_la_SOURCES = theme2k.cpp rotwidget.cpp
ksplash2k_la_LDFLAGS = $(all_libraries) $(KDE_RPATH)
ksplash2k_la_LIBADD = $(LIB_KDEUI) -lksplashthemes

METASOURCES = AUTO

noinst_HEADERS = theme2k.h rotwidget.h

servicesdir = $(kde_servicesdir)
services_DATA = ksplash2k.desktop

themedir = $(kde_datadir)/ksplash/Themes/2k
theme_DATA = Theme.rc Preview.png
```

For more information on writing `Makefile.am` files for KDE, please see the KDE developers' [website](#). The only thing of note is that we provide a default theme based on this plugin, and provide a preview image for it. As a matter of courtesy to your users, you should provide an example `Theme.rc` file illustrating the use of the various options.

Chapter 6

Questions and Answers

This document may have been updated since your installation. You can find the latest version at <http://docs.kde.org/development/en/kdebase/> .

1. *I can't find any themes that work in KSplash. Why is that?*

You probably don't have the correct plugins for the theme. The plugins are in the `kde-artwork` package. Download and install it, and try then again.

2. *What is file `Theme.rc` and how do I make one?*

`Theme.rc` is the file where you can specify a theme's settings. For more information, take a look at [How to make themes for KSplash](#).

Chapter 7

Credits and License

KSplash

Program Copyright (c) 2003 Ravikiran Rajagopal ravi@kde.org

CONTRIBUTORS

- Brian C. Ledbetter brian@shadowcom.net

Documentation Copyright (c) 2003 Teemu Ryttilahti teemu.rytilahti@d5k.net

This documentation is licensed under the terms of the [GNU Free Documentation License](#).

This program is licensed under the terms of the [GNU General Public License](#).

Appendix A

Installation

A.1 Requirements

In order to successfully use KSplash, you need KDE version 3.2 or higher. Some themes may require specific plugins. If a theme does not work, please contact the theme author to find out where to obtain the appropriate plugin.

A.2 Compilation and Installation

In order to compile and install KSplash on your system, type the following in the base directory of the KSplash distribution:

```
% ./configure
% make
% make install
```

Since KSplash uses **autoconf** and **automake** you should have no trouble compiling it. Should you run into problems please report them to the KDE mailing lists.

Appendix B

Source code

B.1 Listing of `theme2k.cpp`

```
#include <qlabel.h>
#include <qwidget.h>

#include <kapplication.h>
#include <kconfig.h>
#include <kdebug.h>
#include <kdialogbase.h>
#include <kgenericfactory.h>
#include <kglobalsettings.h>
#include <klocale.h>
#include <ksplash/objkstheme.h>
#include <kstandarddirs.h>

#include "rotwidget.h"
#include "theme2k.h"
#include "theme2k.moc"

K_EXPORT_COMPONENT_FACTORY( ksplash2k, KGenericFactory< ←
    Theme2k > );

Cfg2k::Cfg2k( KConfig * )
{}

Theme2k::Theme2k( QWidget *parent, const char *name, const ←
    QStringList &args
)
    :ThemeEngine( parent, name, args )
{
    readSettings();
}
```

The KSplash Handbook

```
    initUi();
}

void Theme2k::initUi()
{
    QVBox *vbox = new QVBox( this );
    vbox->setFrameShape( QFrame::WinPanel );
    vbox->setFrameShadow( QFrame::Raised );

    QHBoxLayout *labelBox = new QHBoxLayout( vbox );
    labelBox->setPalette( mTBgColor );
    labelBox->setMargin( 1 );
    QLabel *lbl = new QLabel( mWndTitle, labelBox );
    lbl->setFont( QFont( "Arial", 12, QFont::Bold ) );
    lbl->setPaletteForegroundColor( mTFgColor );

    QLabel *logo = new QLabel( vbox );
    logo->setPalette( Qt::white );

    QString px( locate( "appdata", mTheme->themeDir() +
(mLogoFile.isNull()?QString("/Logo.png"):mLogoFile) ) );
    if (px.isNull())
        px = locate("appdata","Themes/Default/splash_top.png");
    if( !px.isNull() )
    {
        QPixmap pix( px );
        logo->setPixmap( pix );
    }
    else
    {
        logo->setText( "<B>KDE</B>2000" );
        logo->setAlignment( AlignCenter|AlignVCenter );
    }

    mRotator = new RotWidget( vbox, mRotColor1, mRotColor2, ←
        mRotSpeed );

    QHBoxLayout *hbox = new QHBoxLayout( vbox );
    labelBox->setSpacing( 4 );
    labelBox->setMargin( 4 );

    mText = new QLabel( hbox );
    mText->setPaletteForegroundColor( mStatusColor );
    mText->setPaletteBackgroundColor( mTFgColor );
    mText->setText( mWndTitle );
    mText->setFixedHeight( 48 );

    setFixedSize( vbox->sizeHint() );
    QRect rect( KGlobalSettings::splashScreenDesktopGeometry());
    move( rect.x() + (rect.width() - size().width())/2,
```

The KSplash Handbook

```
        rect.y() + (rect.height() - size().height())/2 );
    }

void Theme2k::readSettings()
{
    if( !mTheme )
        return;

    KConfig *cfg = mTheme->themeConfig();
    if( !cfg )
        return;

    cfg->setGroup( QString("KSplash Theme: %1").arg(mTheme->theme()) );

    QColor DefaultTbGColor( Qt::darkBlue );
    QColor DefaultTFgColor( Qt::white );

    mTbGColor = cfg->readColorEntry( "Title Background Color",
&DefaultTbGColor );
    mTFgColor = cfg->readColorEntry( "Title Foreground Color",
&DefaultTFgColor );
    mStatusColor = cfg->readColorEntry( "Status Text Color", &
mTbGColor );

    QColor DefaultRot1( Qt::darkBlue );
    QColor DefaultRot2( Qt::cyan );
    mRotColor1 = cfg->readColorEntry( "Rotator Color 1", &
DefaultRot1 );
    mRotColor2 = cfg->readColorEntry( "Rotator Color 2", &
DefaultRot2 );

    mRotSpeed = cfg->readNumEntry( "Rotator Speed", 30 );
    mWndTitle = cfg->readEntry( "Window Title", i18n("Please
wait...") );
    mLogoFile = cfg->readEntry( "Logo File", QString::null );
}
```

B.2 Listing of rotwidget.h

```
#ifndef __ROTWIDGET_H__
#define __ROTWIDGET_H__

#include <qlabel.h>
#include <qtimer.h>
#include <qwidget.h>
```

The KSplash Handbook

```
#include <kdialogbase.h>
#include <kpixmap.h>

/**
 * @short Display a rotating-gradient widget.
 */
class RotWidget: public QWidget
{
    Q_OBJECT
public:
    RotWidget( QWidget *, const QColor&, const QColor&, int );
    ~RotWidget();

private slots:
    void stepEvent();

protected:
    void preparePixmap( int );
    void paintEvent( QPaintEvent * );
    void resizeEvent( QResizeEvent * );

    QColor m_color1, m_color2;
    int m_step, m_speed;
    QTimer *m_stepTimer;

    QList<KPixmap> m_stepPixmap;
};

#endif
```

B.3 Listing of rotwidget.cpp

```
#include <kdebug.h>
#include <kdialogbase.h>
#include <kpixmapeffect.h>

#include <qlabel.h>
#include <qpainter.h>
#include <qwidget.h>

#include "rotwidget.h"
#include "rotwidget.moc"

RotWidget::RotWidget( QWidget *parent, const QColor& c1, ↵
    const QColor&
    c2, int sp )
```

The KSplash Handbook

```
    :QWidget(parent), m_color1(c1), m_color2(c2), m_step(0), ←
      m_speed(sp)
{
    if( (m_speed <= 0) || (m_speed > 20) )
        m_speed = 1;
    setFixedHeight( 6 );

    for( int i = 0; i <= width(); i++ )
        preparePixmap( i );

    m_stepTimer = new QTimer( this );
    connect(m_stepTimer, SIGNAL(timeout()), this, SLOT( ←
        stepEvent()));
    m_stepTimer->start( 50 );
}

RotWidget::~RotWidget()
{
}

void RotWidget::stepEvent()
{
    // This is inefficient as we create too many pixmaps, ←
    // optimize later.
    m_step += m_speed;
    if( m_step > width() )
        m_step = 0;
    repaint( true );
}

// Todo: Optimize drawing.
void RotWidget::paintEvent( QPaintEvent *pe )
{
    QPainter p;
    p.begin( this );

    QRect r = pe->rect();

    if( m_stepPixmap.at( m_step ) )
        bitBlt( this, r.x(), r.y(), m_stepPixmap.at( m_step ), r. ←
            x(), r.y(),
r.width(), r.height() );
    else
        p.fillRect( rect(), Qt::black );
    p.end();
}

void RotWidget::resizeEvent( QResizeEvent *re )
{
    m_stepPixmap.clear();
}
```

The KSplash Handbook

```
for( int i = 0; i <= re->size().width(); i++ )
    preparePixmap( i );
}

void RotWidget::preparePixmap( int step )
{
    if( step < 0 )
        return;

    // Explicitly draw our first pixmap. The rest we will ↔
    // bitBlt() from here.
    if( step == 0 )
    {
        QPixmap tmp; tmp.resize( size().width() / 2, size(). ↔
            height() );
        QPixmap tmp2(tmp);
        QPixmapEffect::gradient( tmp, m_color1, m_color2,
KPixmapEffect::HorizontalGradient );
        QPixmapEffect::gradient( tmp2, m_color2, m_color1,
KPixmapEffect::HorizontalGradient );
        QPixmap *px = new QPixmap( size() );
        QPainter p;
        p.begin( px );
        p.drawPixmap( 0, 0, tmp );
        p.drawPixmap( size().width()/2, 0, tmp2 );
        p.end();
        m_stepPixmap.append( px );
    }
    else if( m_stepPixmap.at( step-1 ) )
    {
        QPixmap *prev = m_stepPixmap.at( step-1 );
        QPixmap next; next.resize( size() );
        // convert
        // prev = "[-----]"
        // to
        // next = "-----]"
        bitBlt( &next, 0, 0, prev, 1, 0, prev->width()-1, prev-> ↔
            height() );
    };
    bitBlt( &next, width()-1, 0, prev, 0, 0, 1, prev->height ↔
        () );
    QPixmap *n = new QPixmap( next );
    m_stepPixmap.append( n );
}
}
```