Maemo Diablo Reference Manual for maemo 4.1

Porting Software

December 22, 2008

Contents

Chapter 1

Porting Software

1.1 Introduction

The following code examples are used in this chapter:

- maemo-monkey-bubble
- MaemoPad

Much effort has been made in the design of maemo platform to allow easy porting of regular GNU/Linux desktop software to the mobile maemo environment. An earlier chapter in this guide explained the basic tools that ease cross-compilation and help coping with the GNU autotools. This chapter gives pointers to the relevant guides focusing on the differences in the application programming and user interfaces.

Command Line Programs
In most cases, porting software with no user interface is trivial and straightforward. First, the source code of a program is unpacked to the home directory of a Scratchbox user. Second, inside ARMEL target *configure* and *make* are run. Then the compiled program can be tested on the device. Finally the software needs to be packaged.

Programs with Graphical User Interface
The porting of an application that uses GTK+ for its graphical user interface begins the same way as above. In addition to this, the user interface part needs to be refactored to use Hildon instead of directly using GTK+. Dependencies to GNOME components, if any, need to be removed or replaced with corresponding maemo SDK components. If the application uses any components not available in the maemo SDK, these must be also ported by the developer.

Applications that do not use GTK+ and instead use e.g. SDL need more work. The first thing that can be noticed is that the virtual keyboard is missing, because the Hildon Input Method is not available. Thus, application user cannot interact with the application.

Section 1.2 goes into the necessary details of porting an existing GTK+ application to maemo environment.

Localization

The localization of maemo applications is performed using the common *gettext* package. Translating applications to different languages is described in detail in section 1.3.

1.2 Porting Existing Applications to Maemo 4.x

This section describes the process of porting an application to the maemo platform. When starting to port an application to maemo platform, the first step is to set up the development environment. The actual porting after that is described in this section.

The ported game will work on the Scratchbox environment but the hardware key bindings are still missing and therefore it isn't playable on the device yet. For more information about hardware keys can be found in section *Hardware Keys* of chapter *Application Development* in Maemo Diablo Reference Manual.

1.2.1 Introduction

Application that is used as an example for porting is Monkey Bubble, a GNOME/GTK+ based game. It has simple controls, and supports network play. Monkey Bubble version 0.4.0 is used in this example.

The Monkey Bubble interface consists of the main window, menus and a couple of dialogs.

Figure 1.1: Monkey Bubble main window

The figure 1.1 is a screenshot of the main window with an ongoing game. The game is played with configurable keys; the defaults are the arrow keys: left, right and up. The aim is to make all the bubbles disappear. When three or more bubbles with the same color are grouped, they will disappear.

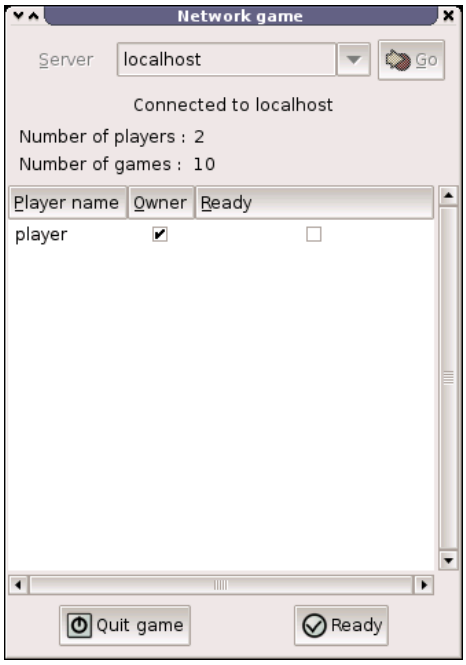


Figure 1.2: Network game window

The figure 1.2 is a screenshot of the network game window, where user can join a network game. It includes a field for selecting server, status about players and game and control buttons.

1.2.2 Application File Structure

Monkey Bubble already has a nicely structured source tree, so there was no need to modify it. XML-formatted Glade [3] files can be found in data directory. Help files can be found in help directory. Graphics can be found in pixmaps directory. Localization files are located in po directory. Audio files can be found in sounds directory. The source code itself is located under src directory in subdirectories, such as audio, input, monkey, net, ui, util and view.

```
./po
./data
./pixmaps
./pixmaps/bubbles
./pixmaps/frozen-bubble
./pixmaps/snake
./pixmaps/number
./sounds
./src
./src/util
./src/input
./src/monkey
./src/view
./src/audio
./src/net
./src/ui
./help
./help/C
./help/fr
```

See also section *Creating Application File Structure* in chapter *Application Development* of the Maemo Diablo Reference Manual.

1.2.3 Requirements and Configure Changes

Monkey Bubble already uses GNU autotools, so the needed Makefile.am and configure.in files are there. Only autogen.sh was missing, so it was created. Sections below discuss the changes needed in Makefile.am and configure.ac files.

```
#!/bin/sh
set -x
glib-gettextize --copy --force
libtoolize --automake
intltoolize --copy --force --automake
aclocal-1.7
autoconf
autoheader
automake-1.7 --add-missing --foreign
```

Listing 1.1: Created maemo-monkey-bubble/autogen.sh

Remember to set permissions to this executable with "chmod a+x autogen.sh".

Maemo SDK meets most of the Monkey Bubble requirements. Biggest exceptions are libgnomeui, bonobo and librsvg. After checking the sources, the libgnomeui and bonobo dependencies can be removed. So, librsvg and its dependencies are additionally needed to be installed.

Start from librsvg installation. One version is available in maemo.org extras repository. Install from there the following packages:

```
librsvg2-2
librsvg2-common
librsvg2-dev
```

Then the configure.in file needs to be modified. The dependency to libgnomeui-2.0 must be removed. And because the UI is going to be hildonized, hildon-1 and libosso need to be added to the list between PKG_CHECK_MODULES(UI,[and following]).

From Makefile.am, remove the help from SUBDIRS list, because the provided help is incompatible with the Hildon help framework. Remove line

help/Makefile from configure.in AC_OUTPUT section as well. Additionally, help can be implemented but it is out of the scope of this section.

After the changes, autogen.sh must be executed to apply the changes.

1.2.4 Basic Porting

Localization and src/ui/main.c

The very first thing is to get Monkey Bubble to work in the maemo environment. For localization, Monkey Bubble uses bonobo. It must be replaced with a compatible localization, described in more detail in section 1.3.

First, localization must be initialized in src/ui/main.c. The following lines must be removed from it:

```
#include <bonobo/bonobo-i18n.h>
#include <libgnomeui/gnome-ui-init.h>
```

Listing 1.2: maemo-monkey-bubble/src/ui/main.c

After that, these lines must be added:

```
#include <locale.h>
#include <libintl.h>
```

Listing 1.3: maemo-monkey-bubble/src/ui/main.c

And new lines in the main() function before bindtextdomain function call:

```
setlocale(LC_ALL, "");
bind_textdomain_codeset(PACKAGE, "UTF-8");
```

Listing 1.4: maemo-monkey-bubble/src/ui/main.c

Remove also the gnome_program_init function call.

For the localization to work, the line #include <bonobo/bonobo-i18n.h> needs to be removed from every file using localization. In order to make the strings localizable, they need to be wrapped in gettext("String") calls. In practice, writing gettext() for every string is tedious. The common practice is to set the following #defines. The N_ is used for gettext_noop(), but it is not available, so the macro does nothing. So the following lines must be added to the beginning of every file using localization:

```
#include <libintl.h>
#define _(String) gettext(String)
#define N_(String) (String)
```

Listing 1.5: maemo-monkey-bubble/src/ui/ui-main.c

Files using localization include the following:

```
src/ui/keyboard-properties.c
src/ui/ui-network-client.c
src/ui/ui-network-server.c
src/ui/ui-main.c
```

Also po/fr.po file needs couple of small fixes. You need to remove `"#, fuzzy"` line from the header and set `Language-Team` to be something else than the default.

Removing GNOME Features

The next step is to remove other GNOME functionality. File src/ui/ui-main.c is next to edit. Remove the following include lines:

```
#include <libgnomeui/gnome-about.h>
#include <libgnome/gnome-sound.h>
#include <libgnome/gnome-help.h>
```

Listing 1.6: maemo-monkey-bubble/src/ui/ui-main.c

Then comment out contents of `show_help_content` and `about` functions, and remove completely function `show_error_dialog` and its prototype from the beginning.

Now the game can be configured and compiled with the following commands:

```
./autogen.sh
./configure --prefix=/usr
make
make install
monkey-bubble
```

Everything should go fine, and the game should be compiled properly. When trying to install and run the game, the following should be seen:

This is fine, but obviously the borders and menus are not hildonized. Also, when trying menu functionality, it is easy to see that the menus will not fit nicely there. So the next step is customization.

1.2.5 User Interface Changes

Hildonizing Main View

Monkey bubble uses Glade [3] for its UI creation. Unfortunately, Glade does not support Hildon, so some changes have to be made.

First of all, in data/monkey-bubble.glade, the main_window type GtkWindow must be changed to GtkVBox. Then all window properties, except "visible" must be removed. After that, the menu must be removed. That is done by removing completely the child containing GtkMenuBar and its subchildren.

Next the src/ui/ui-main.c function ui_main_new should be changed. Add the following lines to the beginning of the function, and remove the KeyboardProperties * kp; line:

```
HildonProgram * program;
GtkWidget * container;
GtkWidget * main_menu;
```

Listing 1.7: maemo-monkey-bubble/src/ui/ui-main.c

Then make the following modifications; the old code is commented out and new lines added after it:

```
#ifdef GNOME
        PRIVATE(ui_main)->window = glade_xml_get_widget( PRIVATE(
            ui_main)->glade_xml, "main_window");
#endif
#ifdef MAEMO
  container = glade_xml_get_widget( PRIVATE(ui_main)->glade_xml, "
      main_window");
  program = HILDON_PROGRAM(hildon_program_get_instance());
  PRIVATE(ui_main)->window = hildon_window_new();
  g_signal_connect_swapped(PRIVATE(ui_main)->window ,"destroy",
      GTK_SIGNAL_FUNC(quit_program),ui_main);
  hildon_program_add_window(program, HILDON_WINDOW(PRIVATE(ui_main)->
      window));
  gtk_container_add(GTK_CONTAINER(PRIVATE(ui_main)->window),
        GTK_WIDGET(container));
  g_set_application_name(_("Monkey Bubble"));
```

Listing 1.8: maemo-monkey-bubble/src/ui/ui-main.c

Because the menu was removed from the Glade file, it must be constructed manually in a Hildon-compatible way. Also, not all the functionality provided by the old menu is needed, so things can be left out. The necessary parts are new game, join network game, pause and quit. Such a tiny menu can be constructed after gtk_box_pack_end function call:

```
main_menu = gtk_menu_new();

item = gtk_menu_item_new_with_label(_("New game"));
g_signal_connect_swapped( item,"activate",GTK_SIGNAL_FUNC(
    new_1_player_game),ui_main);
gtk_menu_append(main_menu, item);

item = gtk_menu_item_new_with_label(_("Join network game"));
g_signal_connect_swapped( item,"activate",GTK_SIGNAL_FUNC(
    new_network_game),ui_main);
gtk_menu_append(main_menu, item);

item = gtk_menu_item_new_with_label(_("Pause"));
g_signal_connect_swapped( item,"activate",GTK_SIGNAL_FUNC(pause_game),
    ui_main);
gtk_menu_append(main_menu, item);

item = gtk_menu_item_new_with_label(_("Quit"));
```

```
g_signal_connect_swapped( item,"activate",GTK_SIGNAL_FUNC(quit_program)
    ,ui_main);
gtk_menu_append(main_menu, item);

hildon_window_set_menu(HILDON_WINDOW(PRIVATE(ui_main)->window),
    GTK_MENU(main_menu));

gtk_widget_show_all(GTK_WIDGET(main_menu));
```

Listing 1.9: maemo-monkey-bubble/src/ui/ui-main.c

After that the code related to the old menu can be removed, starting from the next line and ending to g_signal_connect_swapped function call, the latter being the last removed line. Also the unneeded functions and their prototypes need to be removed:

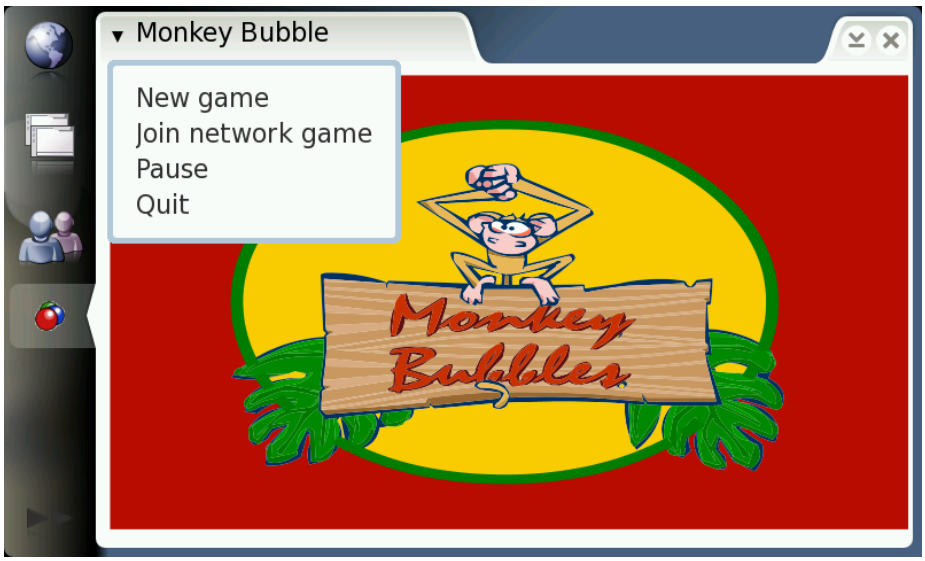
```
ui_main_new_2_player_game
new_2_player_game
new_network_server
stop_game
about
show_help_content
show_preferences_dialog
```

Remember to add the proper include to the beginning of the file:

```
#include <hildon/hildon-program.h>
```

Listing 1.10: maemo-monkey-bubble/src/ui/ui-main.c

Now the main view and menu should be hildonized properly to look like:



Hildonizing Network Window

The Network game window shown in the figure 1.3 is still GtkWindow, and is not hildonized properly. The solution is to make it a GtkVBox and place it under a new GtkDialog.

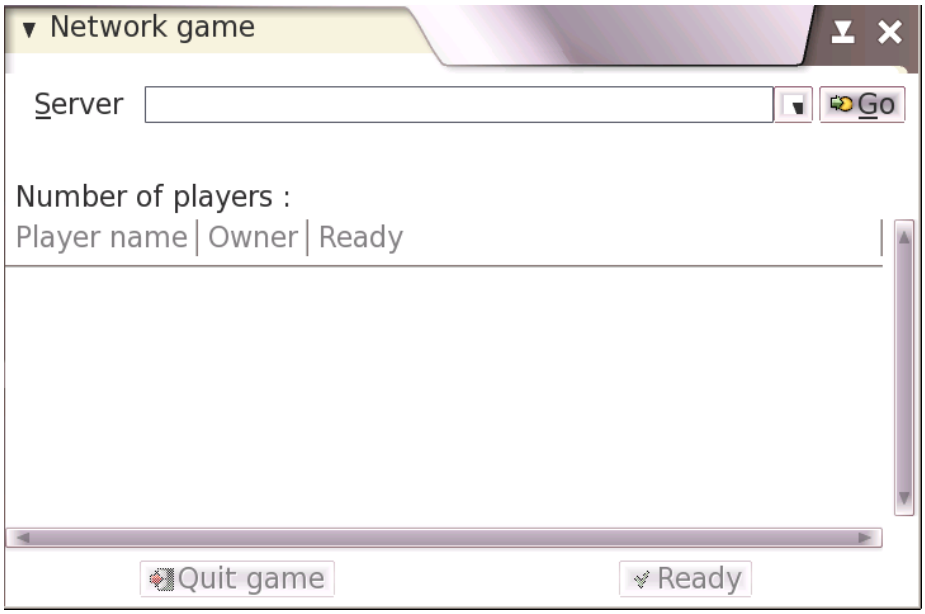


Figure 1.3: Network game window

The starting point here is the glade file located at data/netgame.glade. Change the GtkWindow to GtkVBox and remove all properties except "visible". Change GtkScrolledWindow widget property "height_request" to the value 200 to make everything fit on the screen. A close button is also missing now, because Monkey Bubble relies on the close button of GtkWindow. There is a suitable place between "quit game" and "ready" buttons. You can just copy-paste the whole child containing "quit_button" widget, rename it as "close_button", change the number values of the other widgets name under it for example to 9 to prevent collisions, and change the GtkLabel "Quit game" to "Close". That is enough for the glade file.

Next, the src/ui/ui-network-client.c and ui_network_client_new functions. Add this line to the beginning of the function:

```
GtkWidget * container;
```

Listing 1.11: maemo-monkey-bubble/src/ui/ui-network-client.c

Make the following changes to create a new GtkDialog and show network_window contents under it. The old code is commented out, and the new lines added after it:

```
#ifdef GNOME
        PRIVATE(ngl)->window = glade_xml_get_widget( PRIVATE(ngl)->
            glade_xml, "network_window");
#endif
#ifdef MAEMO
  PRIVATE(ngl)->window = gtk_dialog_new();
  gtk_window_set_title(GTK_WINDOW(PRIVATE(ngl)->window), _("Network
      game"));
  container = glade_xml_get_widget( PRIVATE(ngl)->glade_xml, "
      network_window");
```

```
  gtk_container_add(GTK_CONTAINER(GTK_DIALOG(PRIVATE(ngl)->window)->
      vbox), container);
  item = glade_xml_get_widget( PRIVATE(ngl)->glade_xml, "close_button")
      ;
  g_signal_connect_swapped( item,"clicked",GTK_SIGNAL_FUNC(close_signal
      ),ngl);
#endif
```

Listing 1.12: maemo-monkey-bubble/src/ui/ui-network-client.c

And to the end of the ui_network_client_new function, before return, add the following line:

```
gtk_widget_show_all(GTK_WIDGET(PRIVATE(ngl)->window));
```

Listing 1.13: maemo-monkey-bubble/src/ui/ui-network-client.c

A new function is needed, called close_signal, which is called when the close button is pressed. Add this function before the ui_network_client_new function:

```
static gboolean close_signal(gpointer    callback_data,
                             guint       callback_action,
                             GtkWidget  *widget) {

    UiNetworkClient * self;
    self = UI_NETWORK_CLIENT(callback_data);

    quit_signal(callback_data, callback_action, widget);
    gtk_widget_hide_all(PRIVATE(self)->window);
    return FALSE;
}
```

Listing 1.14: maemo-monkey-bubble/src/ui/ui-network-client.c

There is still one problem: The new close button is located in "connected_game_hbox" widget, which is set insensitive when not connected to the server. The ability to close the window is always needed. The sensitivity of the other widgets has to be changed properly. Add this new function after the previously added close_signal function:

```
void connected_set_sensitive(UiNetworkClient * ngl, gboolean sensitive)
     {
    set_sensitive( glade_xml_get_widget( PRIVATE(ngl)->glade_xml
        ,"scrolledwindow2"), sensitive);
    set_sensitive( glade_xml_get_widget( PRIVATE(ngl)->glade_xml
        ,"quit_button"), sensitive);
    set_sensitive( glade_xml_get_widget( PRIVATE(ngl)->glade_xml
        ,"ready_button"), sensitive);
}
```

Listing 1.15: maemo-monkey-bubble/src/ui/ui-network-client.c

There is one gtk_widget_set_sensitive function call in ui_network_client_new and multiple set_sensitive calls for connected_game_hbox; replace these with:

```
connected_set_sensitive(ngl, FALSE);
```

Listing 1.16: maemo-monkey-bubble/src/ui/ui-network-client.c

There is also one call in recv_network_xml_message function, where the boolean value must be TRUE.

That is all; the network game dialog should now be hildonized and contain a new close button. The "quit game", "ready" and player list should change their sensitivity properly upon connected and disconnected states.

Hildonized network game dialog

1.2.6 State Saving

Maemo supports state saving and background killing application. The application can later be loaded up again with the same state as before. This section describes the steps needed to make Monkey Bubble support state savings.

State Saving Changes in src/ui

Some new files need to be created, first file for global data src/ui/global.h:

```
#ifndef GLOBAL_H
#define GLOBAL_H

#include <libosso.h>

#include "game-1-player.h"

#define MONKEY_TEMP "/tmp/monkey_level_state"

struct GlobalData {
    osso_context_t *osso;
    Game1Player *game;
};

struct StateData {
    int game;
```

```
    int level;
    int score;
    int loadmap;
};

extern struct StateData state;
extern struct GlobalData global;

#endif
```

Listing 1.17: maemo-monkey-bubble/src/ui/global.h

Second one is src/ui/state.h:

```
#ifndef STATE_H
#define STATE_H

#include <glib.h>

gboolean state_load(void);
gboolean state_save(void);
void state_clear(void);

#endif
```

Listing 1.18: maemo-monkey-bubble/src/ui/state.h

Then src/ui/state.c, implementing loading, saving and cleaning the state:

```
#include <libosso.h>

#include "state.h"
#include "global.h"

struct StateData state;

gboolean state_load(void)
{
    osso_state_t osso_state;
    osso_return_t ret;

    osso_state.state_size = sizeof(struct StateData);
    osso_state.state_data = &state;

    ret = osso_state_read(global.osso, &osso_state);
    if (ret != OSSO_OK)
        return FALSE;
    return TRUE;
}

gboolean state_save(void)
{
    osso_state_t osso_state;
    osso_return_t ret;

    osso_state.state_size = sizeof(struct StateData);
    osso_state.state_data = &state;

    ret = osso_state_write(global.osso, &osso_state);

    if (ret != OSSO_OK)
        return FALSE;
    return TRUE;
```

```
}

void state_clear(void)
{
    state.game = 0;
    state.level = 0;
    state.score = 0;
    state.loadmap = 0;

    state_save();
}
```

Listing 1.19: maemo-monkey-bubble/src/ui/state.c

These must be added to src/ui/Makefile.am to monkey_bubble_SOURCES list before $(NULL):

```
state.c state.h global.h \
```

Listing 1.20: maemo-monkey-bubble/src/ui/Makefile.am

Some changes to src/ui/main.c, new includes, definitions and functions:

```
#include <libosso.h>

#include "state.h"
#include "global.h"

#include "game-1-player.h"

#define APPNAME "com.nokia.monkey_bubble"
#define APPVERSION "0.0.1"

struct GlobalData global;

static void _top_cb(const char *args, gpointer data)
{
}

static void _hw_cb(osso_hw_state_t * state, gpointer data)
{
       if(state->shutdown_ind)
       {
               state_save();
               gtk_main_quit();
       }
}

osso_context_t *osso_init(void)
{
    osso_context_t *osso =
        osso_initialize(APPNAME, APPVERSION, TRUE, NULL);

    if (OSSO_OK != osso_application_set_top_cb(osso, _top_cb, NULL))
        return NULL;

    if (OSSO_OK != osso_hw_set_event_cb(osso, NULL, _hw_cb, NULL))
        return NULL;

    return osso;
}
```

Listing 1.21: maemo-monkey-bubble/src/ui/main.c

And change the main() function as follows:

```
  global.osso = osso_init();
  if (!global.osso) {
       perror("osso_init");
       exit(1);
  }
  global.game = NULL;

  /* ... */

  if (!state_load()) state_clear();

  if (state.game == 1) {
       continue_game();
  }

  /* ... */
```

Listing 1.22: maemo-monkey-bubble/src/ui/main.c

Set the topmost callback in src/ui/ui-main.c. It saves the current level and state when Monkey Bubble loses its topmost status, and sets the hibernate flag properly. New continue_game function to continue after loading state:

```
#include "global.h"
#include "state.h"

static void ui_main_new_1_player_game(UiMain * ui_main);

static void ui_main_topmost_cb(GObject *self, GParamSpec *
    property_param, gpointer null)
{
       HildonProgram *program = HILDON_PROGRAM(self);

       if (program == NULL) return;

       if (hildon_program_get_is_topmost(program)) {
               hildon_program_set_can_hibernate(program, FALSE);
       } else {
               if (state.game == 1 && global.game!=NULL) {
                       game_1_player_save(global.game);
                       state.loadmap=1;
               }
               state_save();
               hildon_program_set_can_hibernate(program, TRUE);
       }
}

void continue_game(void) {
        UiMain * ui_main;
        ui_main = ui_main_get_instance();

        ui_main_new_1_player_game(ui_main);
}
```

Listing 1.23: maemo-monkey-bubble/src/ui/ui-main.c

Add `state_clear()` call to program quitting callback:

```
static void quit_program(gpointer    callback_data,
                         guint       callback_action,
                         GtkWidget  *widget) {
```

```
  /* ... */

  state_clear();

  /* ... */
}
```

Listing 1.24: maemo-monkey-bubble/src/ui/ui-main.c

In addition to the ui_main_new() function, after g_set_application_name function call:

```
g_signal_connect(G_OBJECT(program), "notify::is-topmost",
                 G_CALLBACK(ui_main_topmost_cb), NULL);
```

Listing 1.25: maemo-monkey-bubble/src/ui/ui-main.c

To the new_1_player_game function, add before ui_main_new_1_player_game function call:

```
state_clear();
```

Listing 1.26: maemo-monkey-bubble/src/ui/ui-main.c

Function prototype must be added to src/ui/ui-main.h:

```
void continue_game(void);
```

Listing 1.27: maemo-monkey-bubble/src/ui/ui-main.h

Make the following changes in src/ui/game-1-player-manager.c to keep track of level and scores and continue from last level and score:

```
#include "global.h"

/* ... */

static gboolean startnew_function(gpointer data) {

  /* ... */

  PRIVATE(manager)->current_level++;
#ifdef MAEMO
  state.level = PRIVATE(manager)->current_level;
#endif

  /* ... */

}

static void game_1_player_manager_state_changed(Game * game,
                                                Game1PlayerManager *
                                                    manager) {

  /* ... */

    PRIVATE(manager)->current_score = game_1_player_get_score(
        GAME_1_PLAYER(game));
#ifdef MAEMO
    state.score = PRIVATE(manager)->current_score;
#endif
```

```
  /* ... */

}

static void game_1_player_manager_start_level(Game1PlayerManager * g) {

  /* ... */

#ifdef MAEMO
  global.game = game;
#endif

  /* ... */

}

void game_1_player_manager_start(GameManager * g) {

  /* ... */

#ifdef GNOME
  PRIVATE(manager)->current_level = 0;
  PRIVATE(manager)->current_score = 0;
#endif
#ifdef MAEMO
  if (state.game == 1 && state.level > 0 ) {
    PRIVATE(manager)->current_level = state.level;
    PRIVATE(manager)->current_score = state.score;
  } else {
    state.level = 0;
    PRIVATE(manager)->current_level = 0;
    PRIVATE(manager)->current_score = 0;
  }
  state.game = 1;

  /* ... */

}
```

Listing 1.28: maemo-monkey-bubble/src/ui/game-1-player-manager.c

The changes to src/ui/game-1-player.c to save and load the level state:

```
#include "global.h"

/* ... */

void game_1_player_save(Game1Player *game)
{
        monkey_save(PRIVATE(game)->monkey, MONKEY_TEMP);
}

/* ... */

Game1Player * game_1_player_new(GtkWidget * window,MonkeyCanvas *
    canvas, int level,gint score) {

        /* ... */

#ifdef GNOME
        PRIVATE(game)->monkey =
                monkey_new_level_from_file(DATADIR"/monkey-bubble/
                    levels",
```

```
                                                level);
#endif

#ifdef MAEMO
  if (state.loadmap==1) {
          PRIVATE(game)->monkey =
                monkey_new_level_from_file(MONKEY_TEMP, 0);
          state.loadmap = 0;
        } else {
          PRIVATE(game)->monkey =
                monkey_new_level_from_file(DATADIR"/monkey-bubble/
                    levels",
                                                level);
        }
#endif

        /* ... */
}
```

Listing 1.29: maemo-monkey-bubble/src/ui/game-1-player.c

Add function prototype to src/ui/game-1-player.h:

```
void game_1_player_save(Game1Player *game);
```

Listing 1.30: maemo-monkey-bubble/src/ui/game-1-player.h

State Saving Changes in src/monkey

Add save feature to src/monkey/board.c:

```
void
board_save_to_file (Board * board, const char *filename)
{
        #define MUL 4
        GError *error = NULL;
        GIOChannel *channel;
        gint i, j, s=0;
        gchar buffer[COLUMN_COUNT*3+2];
        gsize written = 0;

        if (PRIVATE(board)->bubble_array==NULL) return;

        channel = g_io_channel_new_file (filename, "w+", &error);

        if (channel == NULL) return;

        for (i = 0; i < ROW_COUNT; i++)
        {
                for (j = 0; j < COLUMN_COUNT*MUL; j++)
                        buffer[j] = ' ';

                if (i%2==1) {
                        s = 2;
                } else {
                        s = 0;
                }

                for (j = 0; j < COLUMN_COUNT; j++)
                {
                        Bubble *b;
```

```
                        Color c;

                        if (s>0 && (j+1==COLUMN_COUNT)) break;

                        b = PRIVATE(board)->bubble_array[i*COLUMN_COUNT
                            +j];
                        if (b!=NULL) {
                                c = bubble_get_color(b);
                                buffer[j*MUL+s] = '0'+(int)c;
                        } else {
                                buffer[j*MUL+s] = '-';
                        }
                }
                buffer[COLUMN_COUNT*MUL]='\n';
                buffer[COLUMN_COUNT*MUL+1]=0;

                g_io_channel_write_chars(channel, (const gchar *)&
                    buffer,
                                        -1, &written, &error);
        }

        g_io_channel_shutdown (channel, TRUE, &error);
        g_io_channel_unref (channel);
}
```

Listing 1.31: maemo-monkey-bubble/src/monkey/board.c

And function prototype to src/monkey/board.h:

```
void board_save_to_file (Board * board, const char *filename);
```

Listing 1.32: maemo-monkey-bubble/src/monkey/board.h

Provide a call to this function in src/monkey/playground.c:

```
void
playground_save(Playground * self, const gchar * level_filename)
{
       board_save_to_file(PRIVATE(self)->board, level_filename);
}
```

Listing 1.33: maemo-monkey-bubble/src/monkey/playground.c

And function prototype to src/monkey/playground.h:

```
void playground_save(Playground * self, const gchar * level_filename);
```

Listing 1.34: maemo-monkey-bubble/src/monkey/playground.h

Finally, a call to the save feature in src/monkey/monkey.c:

```
void monkey_save(Monkey *monkey, const gchar * filename)
{
       playground_save(PRIVATE(monkey)->playground, filename);
}
```

Listing 1.35: maemo-monkey-bubble/src/monkey/monkey.c

And function prototype to src/monkey/monkey.h:

```
void monkey_save(Monkey *monkey, const gchar * filename);
```

Listing 1.36: maemo-monkey-bubble/src/monkey/monkey.h

Now the game should be saving its state, and be background killable. It can be tested by setting Monkey Bubble to background (for example by starting another application) and issuing following command:

```
dbus-send --system /com/nokia/ke_recv/bgkill_on com.nokia.ke_recv.bgkill_on.bgkill_on
```

After changing back to Monkey Bubble from Task Bar, it should show banner "Monkey Bubble - resuming" and load pre-background map.

1.2.7 Network Changes

For the networking code, no specific changes to the existing code are needed, but in order for the application to provide a network connection selection dialog or to automatically select a proper connection, the LibConIC library must be used.

The best place for network connection changes is before opening the join network dialog. So, some modifications should be made to src/ui/ui-main.c file. First add a proper header:

```
#include <conic.h>
```

Listing 1.37: maemo-monkey-bubble/src/ui/ui-main.c

After that, modifications are needed in the new_network_game function, which is called when a new network game is launched. Remove the old code and replace it with the following:

```
  UiMain * uimain = UI_MAIN(callback_data);
  PRIVATE(uimain)->ic = con_ic_connection_new();
  g_signal_connect(PRIVATE(uimain)->ic, "connection-event", (GCallback)
      network_connected, NULL);
  con_ic_connection_connect(PRIVATE(uimain)->ic,
      CON_IC_CONNECT_FLAG_NONE);
```

Listing 1.38: maemo-monkey-bubble/src/ui/ui-main.c

The code creates a new ConIcConnection and connects the connection-event signal. This signal is called, when a connection event happens. After that, it calls the connect signal in order to request a new connection. Now, one more thing is needed: to specify what happens, when a connection event comes. That is handled in new network_connected function, which is implemented as follows:

```
static void network_connected(ConIcConnection *cnx,
                              ConIcConnectionEvent *event,
                              gpointer user_data)
{
        UiNetworkClient  * ngl;

        switch (con_ic_connection_event_get_status(event)) {
                case CON_IC_STATUS_CONNECTED:
                  ngl = ui_network_client_new();
                  break;
                case CON_IC_STATUS_DISCONNECTED:
                case CON_IC_STATUS_DISCONNECTING:
                default:
                  break;
        }
```

```
}
```

Listing 1.39: maemo-monkey-bubble/src/ui/ui-main.c

This function simply checks which status the event gives, and if it is CON_IC_STATUS_CONNECTED, it creates a new client window, where the user is able to connect to a server.

There is still one more thing to do. Conic must be added to configure.in into the list following PKG_CHECK_MODULES for UI.

After these changes, starting a new network game will make sure that there is a network connection available.

1.2.8 Integration to Menu

Application integration to menu needs .desktop and .service files. This section describes the needed additions and changes. If you are not familiar with .desktop and .service files, see maemo.org documentation [5] first.

1. Monkey Bubble already has a monkey-bubble.desktop, which is generated from monkey-bubble.desktop.in, so the changes need to be made to it. First of all, the file should be moved under ./data/ directory, in order to make the common maemo application file structure; the path in Makefile.am should be changed accordingly:

   ```
   applications_in_files = data/monkey-bubble.desktop.in
   ```

   And same thing in po/POTFILES.in:

   ```
   data/monkey-bubble.desktop.in
   ```

2. Then the .desktop.in file needs to be modified: The Exec value must have full path, and two new values must be added:

   ```
   Exec=/usr/bin/monkey-bubble
   X-Osso-Type=application/x-executable
   X-Osso-Service=monkey_bubble
   ```

3. Create a service file com.nokia.monkey_bubble.service in directory ./data/:

   ```
   [D-BUS Service]
   Name=com.nokia.monkey_bubble
   Exec=/usr/bin/monkey-bubble
   ```

4. Add section of service file in Makefile.am:

   ```
   dbusdir=$(prefix)/share/dbus-1/services
   dbus_DATA=data/com.nokia.monkey_bubble.service
   ```

5. Change the proper path for desktop file in Makefile.am:

   ```
   applicationsdir = $(datadir)/applications/hildon
   ```

6. These files must be in Makefile.am's EXTRA_DIST variable. The desktop file is there already, so only a service file needs to be added before $(NULL):

   ```
   data/com.nokia.monkey_bubble.service  \
   ```

1.2.9 Application Packaging

This section describes how Monkey Bubble sources were modified to make .deb package building possible.

- Rename Monkey Bubble source directory to maemo-monkey-bubble-0.0.1:

```
mv monkey-bubble-0.4.0 maemo-monkey-bubble-0.0.1
```

- Package source dir maemo-monkey-bubble-0.0.1/ in maemo-monkey-bubble-0.0.1.tar.gz

```
tar czvf maemo-monkey-bubble-0.0.1.tar.gz maemo-monkey-bubble-0.0.1
```

- Go to the source directory

```
cd maemo-monkey-bubble-0.0.1
```

- Set full name environment variable:

```
export DEBFULLNAME="Mr Maemo"
```

Now the actual work can be started. Debian New Maintainers' Guide [2] might be useful in this phase.

- Make initial debianization:

```
dh_make -e xxxxxxx.xxxxxx@maemo.org -f ../maemo-monkey-bubble-0.0.1.tar.gz
```

- Next dh_make will ask a question, and print a summary of the package:

```
Type of package: single binary, multiple binary, library, or kernel module?
 [s/m/l/k] s

Maintainer name : Mr Maemo
Email-Address   : xxxxxxx.xxxxxx@maemo.org
Date            : Thu, 15 May 2008 13:11:58 +0300
Package Name    : maemo-monkey-bubble
Version         : 0.0.1
License         : blank
Type of Package : Single
Hit <enter> to confirm:
```

- Modify debian dir. Only files changelog, compat, control, copyright, docs and rules are needed. In the control file, packages `libhildon1`, `librsvg2-2` and `libglade2-0` must be added to Depends. You should also set the Section field into user/games to make the package compatible with Application Manager. After that, the package structure is ok. Then the configuration files need to be changed to be able to build the package.

  N.B. libglade2 must be installed from Maemo repositories [4]. ARMEL Debian packages of librsvg2-2 and librsvg2-common are available from maemo.org extras repository.

- Because the key definition dialog was disabled, definitions need to be set in debian/postinst:

```
#!/bin/sh

set -e
```

```
case "$1" in
    configure)

    gconftool-2 -s /apps/monkey-bubble/player_1_shoot --type=
        string Up
    gconftool-2 -s /apps/monkey-bubble/player_1_left --type=string
         Left
    gconftool-2 -s /apps/monkey-bubble/player_1_right --type=
        string Right
    ;;

    abort-upgrade|abort-remove|abort-deconfigure)

    ;;

    *)
        echo "postinst called with unknown argument \`$1'" >&2
        exit 1
    ;;
esac

exit 0
```

Listing 1.40: maemo-monkey-bubble/debian/postinst

It should be set executable with "chmod a+x debian/postinst".

- A link to the desktop file must be added to /etc/others-menu/extra_applications to show it in menu. The link is created by making a debian/maemo-monkey-bubble.links file with following contents:

```
usr/share/applications/hildon/monkey-bubble.desktop
etc/others-menu/extra_applications/monkey-bubble.desktop
```

- Add the file maemo-monkey-bubble.files to list all the files that will be installed by the package.

- Add the following lines in configure.in:

```
PKG_CHECK_MODULES(HILDON, hildon-1 >= 0.14.8)
AC_SUBST(HILDON_LIBS)
AC_SUBST(HILDON_CFLAGS)
```

- Make the following two changes in Makefile.am (remember to use tab to indent):

  Add items in EXTRA_DIST before $(NULL):

```
        autogen.sh \
        debian/changelog \
        debian/compat \
        debian/copyright \
        debian/control \
        debian/rules \
        debian/docs \
```

  Add deb rule:

```
deb: dist
        -mkdir $(top_builddir)/debian-build
        cd $(top_builddir)/debian-build && tar zxf \
../$(top_builddir)/$(PACKAGE)-$(VERSION).tar.gz
        cd $(top_builddir)/debian-build/$(PACKAGE)-$(VERSION) && dpkg-buildpackage \
-rfakeroot
        -rm -rf $(top_builddir)/debian-build/$(PACKAGE)-$(VERSION)
```

Now the source dir should be ready for packaging, described in section [1].

If you want to create a package that is usable with the Application Manager, see section [6].

1.3 Maemo Localization

1.3.1 Overview

This section describes how to localize maemo applications. Localization is needed to provide native translations of the software. The section contains information on how to localize an application, how to ease extraction of message strings, how to make the translations and how to test the localization.

Changes are presented with code examples to help the localization process. MaemoPad is used as an example here.

1.3.2 Localization

Localization means translating the application to different languages. Maemo localization is based on the standard gettext package, and all the necessary tools are included in Scratchbox. For applications, localization requires a couple of files in the po/ directory. The following files will be used for localization:

```
Makefile.am
en_GB.po
POTFILES.in
```

POTFILES.in contains the list of source code files that will be localized. File *en_GB.po* includes translated text for target en_GB.

1.3.3 Localizing Application

To localize an application, l10n needs to be set up before gtk_init() by including the following headers.

```
#include <libintl.h>
#include <locale.h>
```

Listing 1.41: maemopad/src/main.c

To initialize the locale functions, the following lines need to be added:

```
setlocale(LC_ALL, "");
bindtextdomain(GETTEXT_PACKAGE, LOCALEDIR);
bind_textdomain_codeset(GETTEXT_PACKAGE, "UTF-8");
textdomain(GETTEXT_PACKAGE);
```

Listing 1.42: maemopad/src/main.c

The most important of these are GETTEXT_PACKAGE and LOCALEDIR, which come from configure.ac:

```
localedir=/usr/share/locale
AC_PROG_INTLTOOL([0.23])
GETTEXT_PACKAGE=maemopad
AC_SUBST( GETTEXT_PACKAGE )
ALL_LINGUAS="en_GB"
AM_GLIB_GNU_GETTEXT
```

Listing 1.43: maemopad/configure.ac

Of these, only GETTEXT_PACKAGE needs special attention, as it is the l10n domain that is to be used, and ALL_LINGUAS lists the available translations.

1.3.4 Easing Extraction of Strings

In the source code, there are multiple strings that eventually get shown in the user interface. For example:

```
g_set_application_name ( _("MaemoPad") );
```

Listing 1.44: maemopad/src/main.c

In order to make the strings localizable, they need to be wrapped in gettext("String") calls. In practice, writing gettext() for every string is tedious. The common practice is to set the following #define

```
#define _(String) gettext (String)
```

Listing 1.45: maemopad/src/main.c

Thus, the i18n version of the example would be:

```
hildon_app_set_title ( app, _("MaemoPad") );
```

Creating Translation Files

Creating *.po files is quite straightforward. Maemopad only has one localization file, *en_GB.po*; others, such as *fi_FI.po*, can be easily made based on this. Localization .po files are filled with simple structures, defining the localization id and the actual text string, e.g.

```
#: src/document.c:727 src/document.c:733
msgid "note_ib_autosave_recover_failed"
msgstr "Recovering autosaved file failed"
```

"msgid" defines the original string (key) used in code, and "msgstr" defines the translated string for localization.

First, a template file is created with all strings from sources for translation. GNU xgettext command is used to extract the strings from sources:

```
xgettext -f POTFILES.in -C -a -o template.po
```

Option "-f POTFILES.in" uses POTFILES.in to get the files to be localized, "-C" is for C-code type of strings, "-a" is for ensuring that all the strings are received from sources, and "-o template.po" defines the output filename.

This may output some warnings. Usually they are not serious, but it is better to check them anyway.

If the translation in question is into Finnish, the edited *po/template.po* needs to be copied to *po/fi_FI.po*, and *fi_FI.po* needs to be added to ALL_LINGUAS in *configure.ac*.

Now *po/fi_FI.po* will include lines such as the following:

```
#: ../src/ui/interface.c:123
msgid "maemopad_save_changes_made"
msgstr "Save changes?"
```

All msgstrings should be translated into the desired language:

```
#: ../src/ui/interface.c:123
msgid "maemopad_save_changes_made"
msgstr "Tallenna muutokset?"
```

Autotools should now automatically convert the .po file to .mo file during the build process, and install it to the correct location.

To do it manually:

```
msgfmt po/fi_FI.po -o debian/maemopad/usr/share/locale/fi_FI/LC_MESSAGES/domain.mo
```

Where "debian/maemopad/usr/share/locale/fi_FI/LC_MESSAGES/" is the directory where the software should be installed to.

Testing

Inside scratchbox, the translated application should be tested by setting LC_ALL environment variable to contain the newly created locale (in this example case for Finnish translation, "fi_FI"):

```
LC_ALL="fi_FI" run-standalone.sh ./maemopad
```

On the target device, a wrapper script is needed to accomplish this, if the device does not have a locale for the locale identifier used. The script can simply consist of:

```
#!/bin/sh
LC_ALL="fi_FI" /usr/bin/maemopad
```

The example above assumes that the installation package installs the executable into /usr/bin. This script needs then to be configured to be run in the application's .desktop file, instead of the actual executable.

Bibliography

[1] Maemo Diablo Reference Manual for maemo 4.1, chapter Packaging, Deploying and Distributing, section *Creating Debian Packages*. http://maemo.org/development/documentation/.

[2] Debian New Maintainers' Guide. http://www.debian.org/doc/manuals/maint-guide/.

[3] Glade User Interface Builder. http://glade.gnome.org/.

[4] Maemo repository for libglade2. http://repository.maemo.org/pool/diablo/free/libg/libglade2/.

[5] Maemo developer documentation. http://maemo.org/development/documentation/.

[6] Maemo Diablo Reference Manual for maemo 4.1, chapter Packaging, Deploying and Distributing, section *Making Application Packages*. http://maemo.org/development/documentation/.