

Understanding
ACCESS Linux Platform™
Application
Development



ACCESS™

ACCESS Systems Americas
Bill Lee
Developer Relations Engineer



Topics

1. Practical Elements of the Application
2. Essential Concepts of the App Lifecycle

Elements of an
ACCESS Linux Platform™
Application

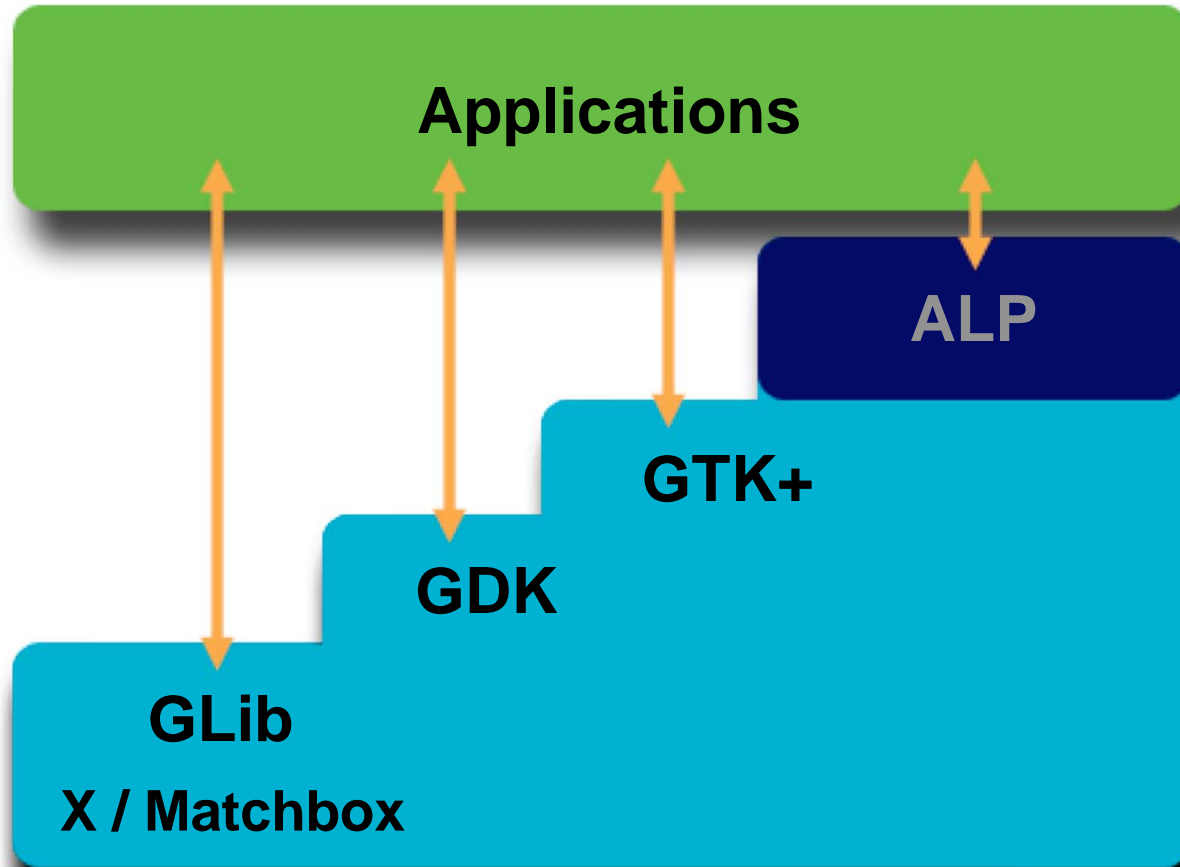




Topics: Practical Elements of the Application

- Application Model
- Working with the Application Framework
- Elements of an Application

X, Glib/GDK/GTK+, "ALP"



C++ Issues

- Feel Free to Use C++
- GTKmm – *not recommended*
- libstdc++ – *try to avoid*
- **gcc** vs. **g++**
- libalp_stdcc++ vs. libstdc++
 - Small Footprint
 - No RTTI
 - No Exceptions
 - No iostream

C Application

The "Minimalist" Application

```
int main( int argc, char *argv[] )  
{  
  
    return 0;  
} // main()
```

GTK+ Minimal Program with an Invisible UI

```
#include <gtk/gtk.h>           // GTK
gint main(gint argc, gchar *argv[] )
{
    gtk_init(&argc, &argv); // Init the GTK GUI
    gtk_main();             // Run
    return 0;
} // main()
```

Manifest File

- Contains Descriptive Application Meta-data
- Loaded by “Bundle Manager”
- Add Application Specific Tags, Accessible via “Bundle Manager”



Manifest Sample

```
<manifest name="com.mydomain.apps.Hello">
  <application>
    <name>HelloGlade</name>
    <icon>app.png</icon>
  </application>
</manifest>
```

- Required Tags/Attributes
 - `<manifest name="bundle_name">` - Root Tag
 - `<name>visible_name</name>` - Launcher Name
 - `<icon>icon_file</icon>` - Launcher Icon

Application Module Structure

- Packaged as “**Bundle AR**chive” Single-file *bundle_name.bar*
 - Compressed file == Expanded directory
 - “Executable” build as a shared object **libalp_name.so**
 - Icons, Localization Files, Graphics, etc.
- Installed in Device (or Simulator) Directory
 - **/opt/alp/bundles**
 - **/var/opt/alp/bundles**
 - Removable Media
 - Licensee Defined Areas

Changes to GTK Program Source

- `main()` → `alp_main()`
- Add Exit Callback
`alp_app_add_exit_handler(myexit, NULL);`
- User Interface Changes
 - Use **AlpTitleBar** and **AlpMenuBar** rather than Window title and **GtkMenu**
 - Eliminate User “Quit”

Minimalist C Application

```
int    main( int argc,  char *argv[] )  
{  
  
    return 0;  
} // main()
```

Minimalist GTK+ Application

```
#include <gtk/gtk.h>

gint      main(gint argc, gchar *argv[])
{

    gtk_init(&argc, &argv); // Init the system

    gtk_main();             // Run

    return 0;
} // main()
```

Minimalist ALP Application

```
#include <gtk/gtk.h>
#include <alp/alp.h>

gint alp_main(gint argc, gchar *argv[])
{
    // Hook the ALP exit handler
    alp_app_add_exit_handler(ExitTheApp, NULL);

    gtk_init(&argc, &argv); // Init the system

    gtk_main();           // Run

    return 0;
} // main()
```

Exit Handler

- No User Quit – System Generated
- System Calls to Signal Termination of App

```
// Application exit handler  
void ExitTheApp(gpointer cbData)  
{  
    gtk_main_quit();  
} // ExitTheApp()
```

Lab 1: Experimenting with the App

1. Open Project "LW07 – Workshop 1"
2. Create Exit Handler
 1. Call `gtk_main_quit()`
 2. Add `g_print()`
3. Set Exit Handler
`alp_app_add_exit_handler()`
4. Test Application Behavior
 1. Create function to do Some work – return **true**
 2. Add via `gtk_add_idle()` to run function
 3. Hit the [**home**] key and see what happens
 4. See what happens when the exit handler is disabled

User Interface Considerations

- No User Initiated "Quit"
- Use AlpTitleBar/AlpMenuBar not Window Title and GtkMenu
- Do Not Rely on Specific UI Element Sizes
- Do Not Layout to Specific Pixel Offsets
- Theming is Your Friend...

User Interface Theming

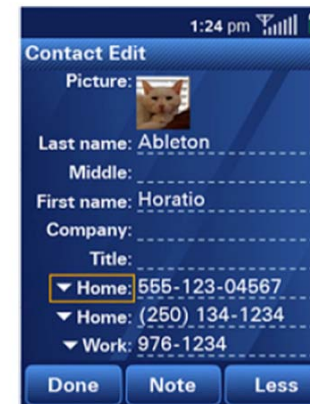
- Carrier Customization
- User Personalization
- Handset Vendor Customization
- GTK+ Theming Provides Basis for ALP Theming—Extensions & Tools



Wireframe



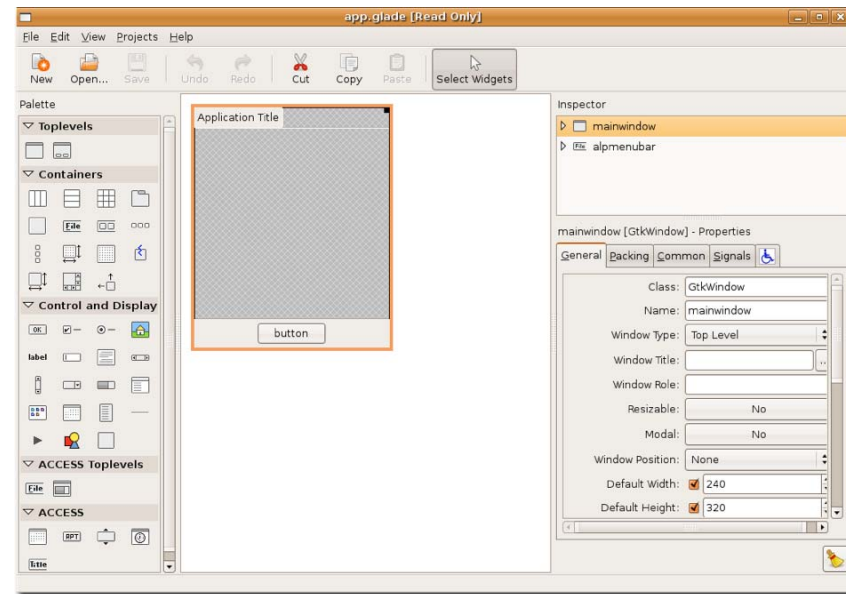
Un-themed



ALP Themed

Building User Interfaces with Glade

- Open/Create **.glade** XML file with **alp-glade**
- Save **.glade** file in **rsc** directory
- Include **glade/glade.h**
- Update **Makefile**
- Link with **libglade**



Bind “Presentation” to Application

- GTK+ APIs Used to Program the Interface
- Libglade APIs Used Interrogate the Glade Structure
 - Bindings to Signal Handlers Automatic
 - Elements of UI Referenced by Glade Widget Name

```
gtk_init();  
GladeXML *glade =  
    alp_bundle_acquire_glade_xml("app.glade",  
    NULL);  
  
    // ... Initialize User Interface Elements ...  
glade_xml_signal_autoconnect(glade);  
gtk_main();
```

Lab 2: Creating the User Interface

1. Open Project "LW07 – Workshop 2"
2. Open existing **.glade** file
3. Add Vertical Box (**GtkVBox**) w/3 sections
4. Add and Configure **AlpMenuBar**
5. Drop **AlpTitleBar** into Top Section
Set Title Bar Menu Property to Refer to **AlpMenuBar**
6. Drop Horizontal Button Bar (**GtkHButtonBox**) into bottom section of Vertical Box
7. Drop Scroll Window (**GtkScrolledWindow**) into mid-section
8. Add libglade code to load **.glade** file
 - **alp_bundle_acquire_glade_xml()** (not **glade_xml_new()**)
 - **glade_xml_signal_autoconnect()**

Essential Concepts of the
ACCESS Linux Platform™
Application Lifecycle





Topics: Essential Elements of the App Lifecycle

- Understanding for Mobile Device Usage
- Application Lifecycle
- Launch Codes
- ALP Application Template

Mobile Device Constraints

Fewer Resources than Desktop

- Smaller Screen
- Less Memory
- Less Storage
- Slower Processor
- Power Constraints



Task-oriented User Model

- Many Short Tasks
- Long Tasks – Interrupted
- One Task at a Time
- Not Aware of “Applications”

Key Concepts

1. Application Lifecycle
 - System ends app, not user
 - An app ends when new app starts
2. Launch Codes Interrupt App for System Interactions

Application's Runtime Impact

- Run → Active → Paused → Active → Paused → ...
- Invocation → User Interface → Interruption → Re-
invoke → Interruption → ...
- Application Usage Time ≠ Process Life
- Perception of Continuous Execution

Application Process Flow

- “Primary” Invocation
 - Full Use of Display
 - User Input
- Single Application Instance
- Exits When Another Application Becomes “Primary”
- No Explicit User “Quit”

Application Process Flow

Primary Invocation with Exit

- Single Application Instance
- Exits When Another Application Becomes "Primary"

1. Invocation
2. →Run (Display)
3. →Interruption: Exit
4. Other App Runs as "Primary"
5. User Re-invokes First App →1.

Save/Restore Application State

- Perception of Continuous Execution
- Be Prepared for App to End at Any Time
- Exit Handler
- Save Application State Before Exit
 - UI State Can be Saved During Window Destruction
 - Bundle Manager APIs to Retrieve Writable File Path
 - Save App State in Persistent Storage (File, Global Settings, etc.)
- Restore Application State
 - Bundle Manager APIs to Retrieve File Path
 - Initialize UI State After Reconstructing UI

Lab 2.5: Basic ALP Application

1. Open Project "LW07 – Project 2-solution"
2. Add **onInitializeApp()** function
3. Add **onTerminateApp()** function
4. Now a complete simple application!

Launch Codes

- Invocation → User Interface → Interruption
→ Re-invoke → Interruption → ...
- Application Usage Time \neq Process Life
- Perception of Continuous Execution:
Save/Restore

Launch Code Invocations

Interrupts Application with "System" Events

- `ALP_APP_PRIMARY`
- `ALP_APP_ALARM`
- `ALP_APP_ALARM_REASON`
- `ALP_APP_FIND`
- `ALP_APP_FIND_CANCEL`
- `ALP_APP_DISPLAY`
- `ALP_APP_NOTIFY`
- `ALP_APP_BACKGROUNDED`
- `ALP_APP_EXCHANGE`
- `ALP_APP_TRANSIENT`

Launch Codes

- Received as *argc* and *argv*
- Launch Codes are Strings Beginning with "**--alp-...**"
 - Use **ALP_APP_...** Instead
 - Some Launch Codes Have Parameters
 - More than One Launch Code can be Sent

Relaunch Handler

How Does the System Send Launch Codes?

- If Process is not Active: **alp_main()** is called
- If Process is Active: Call "Relaunch Handler"
alp_app_set_relaunch_handler()

- Is App running?
 - No: Call **alp_main()**
 - Yes: Is Relaunch handler registered?
 - No: Shutdown app, then call its **alp_main()**
 - Yes: Call its relaunch handler

Application Must be Responsive

- Application Must Respond to
 - Request to Exit
 - Accept a Launch Code
- Application Responds with
 - Exit
 - Return to Primary Event Loop
- If No Response within 3 seconds, the App is Terminated



Launch Codes

- Primary Display
- Notifications
- Alarms
- Alerts
- Searching
- Backgrounding

Notifications

Provides Mechanism for Notifying Apps of Specific Unsolicited Events

- Boot, Install
- Global Settings
- Attention Manager
- Bundle Manager
- Volume Manager
- Telephony SIM, Data, Voice, Network, Phonebook
- Date and Time Changes
- Power Manager

Persistent vs. Temporary Notifications

- Apps Register to Receive Specific Notifications
- Temporary Notifications
 - Only active when application is running
 - Handled through function callbacks
 - Efficient and minimal system overhead
- Persistent Notifications
 - Can be received whether application is active or not
 - Received as **ALP_APP_NOTIFY** Launch Code
 - Can register during installation (persists across reboots)
 - Automatically unregistered when app uninstalled

Initialization Notification

- Install/Registration-time Initialization
 - Update Manifest file with
`<notifications><register/></notifications>`
- For One-time Initialization
 - Initializing file data
 - Initializing Global settings
 - Initializing SQL data
 - Registering for persistent notifications
 - Registering Exchange Manager handling
- No Need for Boot-time Notification Handling

Alarms

- “Clock-time” Application Response
- Application Defines an Alarm by ID
- Received as a **ALP_APP_ALARM** Launch Code
- Redefine Alarm Setting by Using Existing ID
- No Display During Alarm Handling – Use Alerts
- **`alp_alarm_set(appID, ref, seconds)`**

Attention Manager and Alerts

- Mechanism to Inform the User of Something “Significant”
 - e.g. Incoming Call, Time for an appointment has arrived, Urgent Email, Target Stock Price, Battery Low
- Manages Presentation of Items Needing Attention
- Uses Priority Scheme
 - Background applications can display dialogs
 - System alerts displayed in front of all other windows

Alerts

- Application Posts Alert with an App-Specific ID String
- If User Selects an Alert, Application Receives **ALP_APP_DISPLAY** Launch Code with ID String
- Custom Options to Specify Look and Function of Alert Dialog
- **alp_attn_alert_post**(*sourceAppId*,
alertTypeName, *handle*, *interface*, *priority*,
duration, *nprops*, *properties*[])

Template-based Application

- Use Template to Prepare for Launch Code Handling
- Focus Efforts on Application Solution

Lab 4: Alarms and Alerts

1. Open "LW07-Project3-Template"
2. Open **app.glade** file
3. Change button to set Alarm
 1. Retitle button
 2. Set signal for button
 3. Add signal handler
4. Open **myapp.c**
5. Add signal handler
 1. `alp_alm_set_alarm()`
 2. `alp_bundle_name(alp_bundle_application())`

Lab 4: Alarms and Alerts (continued)

6. Add **onAlarm()** Handling: Post "Alert"
 1. Set up Properties
 2. **alp_attn_alert_post()**
7. Add **onFindAlertDisplay()** Handling
 1. Prepare application for display to respond to alert
 2. Process **ALP_APP_PRIMARY**

Searching via "Find"

- Supports User Instigated Search of App Data
- User Interface Presented by System
- Handle Find Launch Code (**ALP_APP_FIND**)
- Application Informs System of Items Found
 - Identifies items by an Application specific String ID
- User Selects Item from System UI
- System Sends String ID to Application (**ALP_APP_DISPLAY**)
- User May Cancel Searching (**ALP_APP_FIND_CANCEL**)

Find Handling

- Add `<FindVersion>1.0</FindVersion>` to Manifest

ALP_APP_FIND

- `alp_find_results_start()`/
`alp_find_results_end()`
- Loop over data
- Call `alp_find_results_submit_one()`
 - User description and app specific string ID
- Do not spend longer than 3-secs.

ALP_APP_FIND_CANCEL

- Terminate Searching

Displaying User Selections

ALP_APP_DISPLAY

- Sent in Reponse to User Selection of Find or Alert Entry
- App Specific String ID Submitted via **alp_find_results_submit_one()** or **alp_attn_alert_post()**
- Prepare to Display User's Selected Item
 - Do not actually display – **ALP_APP_PRIMARY** is also sent

Backgrounding

- A Way to Run in the “Background,” but...
- Not What You’d Expect
(due to 3-second limitation)
- Reasons for Backgrounding
 - Extended application termination
 - Perform Event based extended processing (via event loop)
 - Monitor secondary thread that performs background processing

Application Process Flow

Primary Invocation with Exit

- Single Application Instance
- Exits When Another Application Becomes "Primary"

1. Invocation

2. → Run (Display)

3. → Interruption: Exit

Application Process Flow

Primary Invocation without Exit: Backgrounded

- Single Application Instance
- Continues When Another Application Becomes "Primary"

1. Invocation
2. →Run (Display)
3. →Interruption: Hide
4. Other App Runs as "Primary"
5. User Re-invokes First App (Already Running) →1.

Handling Background Launch Code

- Add to Manifest:
<backgrounding>setting</backgrounding>
 - Unsupported
 - Supported
 - Preferred
 - Required
- Note: App Can Still be Requested to Exit

Background Operation

- Handle **ALP_APP_BACKGROUNDED** Launch Code
- Close All Dialogs
(**alp_max_dialog_cancel_all()**)
- Should Hide or Destroy User Interface
- Terminate App When Background Processing is Done
- Be Prepared for Other Launch Codes Including **ALP_APP_PRIMARY**

- If Operation > 3-seconds, Use Event Loop

Template-based Application

- Use Template to Prepare for Launch Code Handling
- Focus Efforts on Application Solution
- Consolidated Application State (No Globals)
- Main – Templated Application Flow
 - Launch Code handling
 - Backgrounding and Find handling pattern
- Myapp – Application Specific Implementation
 - Implement “callbacks” from main
- Apputils – Optional Support Utilities
 - Some insulation from API changes

Template Organization

- See the “LW07-Project3-Template” Project
 - **Manifest.xml, Makefile, rsc, src**
- Main – Templated Application Flow
 - Application Lifecycle Management
 - Launch Code handling
 - Backgrounding and Find handling pattern
- Myapp – Application Specific Implementation
 - Implement “callbacks” from main
- Apputils – Optional Support Utilities
 - Some insulation from API changes

Template Code: myapp.c/.h

- Add "Global" Data to **UserData_t**
- Implement Generic Flow Functions
 - **doInitializeApp()** – Restore application state
 - **doTerminateApp()** – Save application state
 - **doAllocUserData()** – Add any UserData pre-initialization
 - **doInitializeUI()** – Initialize user interface (or Glade Elements)
 - **doInstallNotification()** – Install time initialization
- Implement Launch Code Functions
 - **onAlarm()** – **ALP_APP_ALARM** handling
 - **onBackground()** – **ALP_APP_BACKGROUNDED** event loop
 - **onFind()** – **ALP_APP_FIND** event loop
 - **onFindAlertDisplay()** – **ALP_APP_DISPLAY** handling
 - **onExchange()** – **ALP_APP_EXCHANGE** handling
 - **onNotification()** – **ALP_APP_NOTIFY** handling



Eat, drink and
GO NATIVE!
Grab your swizzle sticks and
join ACCESS for the kick-off
party of LinuxWorld 2007

Network and enjoy delicious “native-inspired”
food, drinks and entertainment

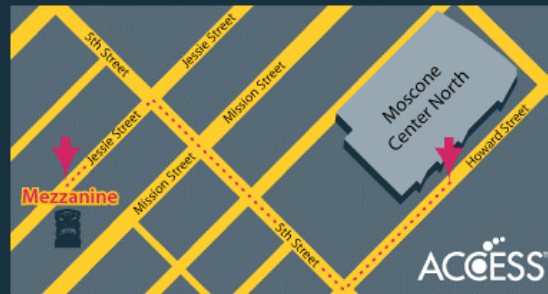
When: Tuesday, August 7th - 6:00 p.m. - 9:00 p.m.

Where: Mezzanine - 444 Jessie Street (just 3 blocks from Moscone)

Getting There: FREE shuttle service provided to and from Moscone Center starting at 5:30 p.m.

First pick-up at 5:30pm - outside Moscone Center.

Final pick-up at 9:15pm - outside Mezzanine.



Bibliography

- *The Official GNOME 2 Developer's Guide*, by Matthias Warkus, published by No Starch Press
- GLib, GDK, GTK+
 - `http://www.gtk.org/tutorial`
 - `http://developer.gnome.org/doc/API`
 - `.../2.0/gtk`
 - `.../2.0/gdk`
 - `.../2.0/glib`
 - `.../glib-Message-Logging.html`
 - `.../2.0/gobject`
 - `.../2.0/libglade`
- `http://accessdevnet.com`



Join ACCESS Developer Network (ADN)

- How do Developers Participate?

<http://accessdevnet.com>

- Access to the Resource Pavilion
- Apply Early ACCESS Program



Thank You

ACCESS Systems Americas
Bill Lee
Developer Relations Engineer

