



Understanding ACCESS Linux Platform™ Application Development



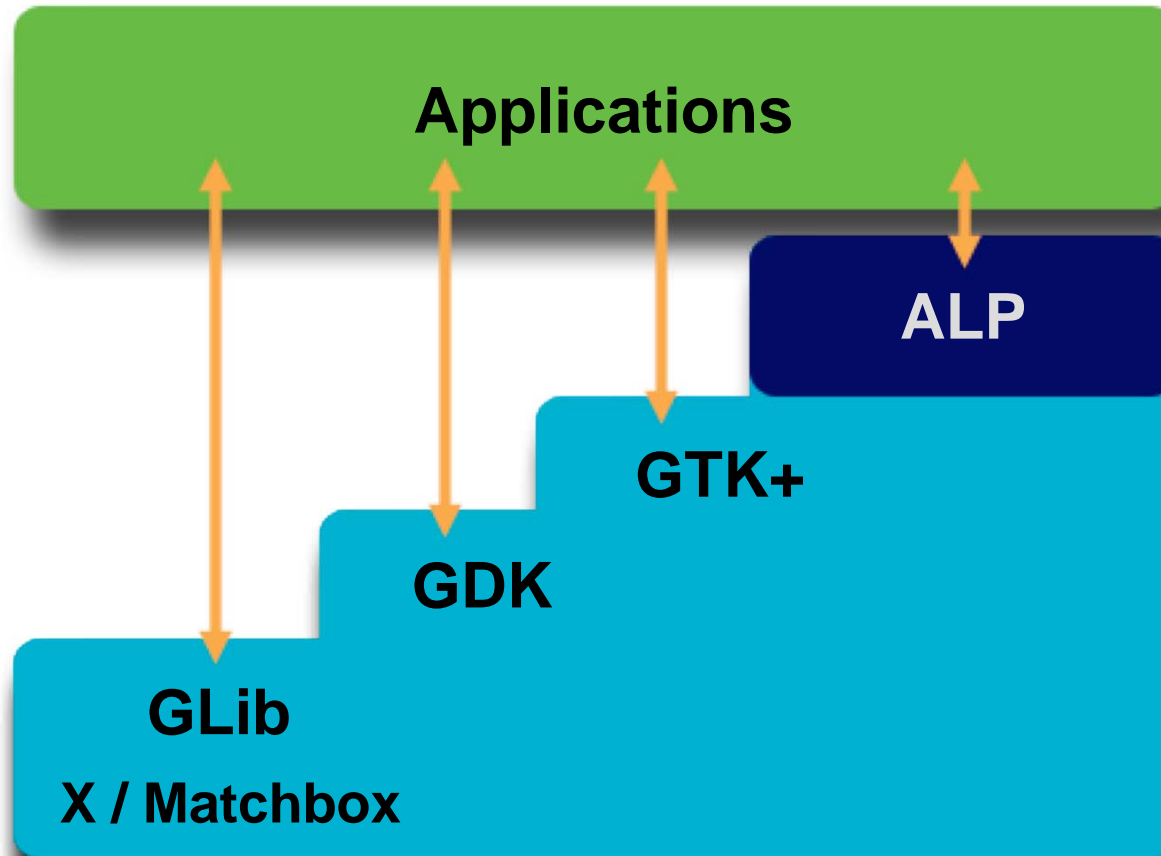
ACCESS Systems Americas
Bill Lee
Developer Relations Engineer
bill.lee@access-company.com

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



Elements of an ACCESS Linux Platform™ Application

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



- We Provide C Interface
- Feel Free to Use C++
- “Constrained” Device ... Avoid
 - GTKmm
 - RTTI
 - Exceptions
 - iostream

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 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

○ Packaged as “**Bundle ARchive**” Single-file

bundle_name.**bar**

- Compressed file == Expanded directory
- “Executable” build as a shared object
- Icons, Localization Files, Graphics, etc.

○ Installed in Device (or Simulator) Directory

- `/opt/alp/bundles`
- `/var/opt/alp/bundles`
- Removable Media
- Licensee Defined Areas

- `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()
```

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

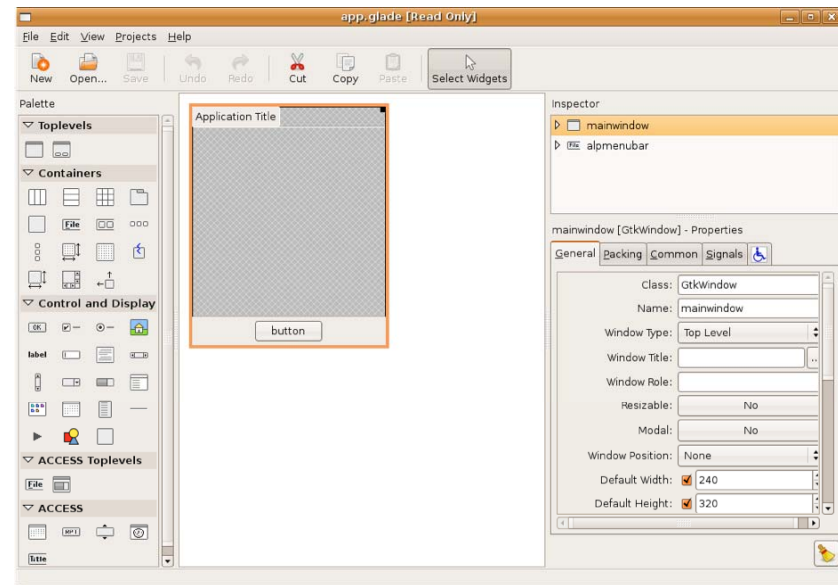
```
// Application exit handler  
void ExitTheApp(gpointer cbData)  
{  
    gtk_main_quit();  
} // ExitTheApp()
```

1. Open Project “OSiM07-Project1”
2. Create Exit Handler
 - a. Call `gtk_main_quit()`
 - b. Add `g_print()` or `APP_TR()`
3. Set Exit Handler
`alp_app_add_exit_handler()`
4. Test Application Behavior
 - a. Create function to do Some work – return `true`
 - b. Add via `gtk_add_idle()` to run function
 - c. Hit the [`home`] key and see what happens
 - d. See what happens when the exit handler is disabled

- 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...

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 “OSiM07-Project2”
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

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”

1. Application Lifecycle

- System ends app, not user
- An app ends when new app starts

2. Launch Codes Interrupt App for System Interactions

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

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

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.

- 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 3: Basic ALP Application



1. Open Project “OSiM07-Project3”
2. Add `onInitializeApp()` function
3. Add `onTerminateApp()` function
4. Now a complete simple application!

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

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

- 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

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

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

- 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

- 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

- “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)`

- 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

- 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, stringID, 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 “OSiM07-Project4-Template”
2. Open `app.glade` file
3. Change button to set Alarm
 - a. Retitle button
 - b. Set signal for button
 - c. Add signal handler
4. Open `myapp.c`
5. Add signal handler
 - a. `alp_alm_set_alarm()`
 - b. `alp_bundle_name(alp_bundle_application())`

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**)

- 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

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

- 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

Primary Invocation with Exit

- Single Application Instance
- Exits When Another Application Becomes “Primary”

1. Invocation

2. →Run (Display)

3. →Interruption: Exit

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.

- Add to Manifest:
`<backgrounding>setting</backgrounding>`
 - Unsupported
 - Supported
 - Preferred
 - Required

- Note: App Can Still be Requested to Exit

- 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

- 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

- See the “OSiM07-Project4-Template” Project
 - `Manifest.xml`, `Makefile`, `rsc`, `src`
 - Doxygen HTML Documentation: `doxygen/index.html`
- 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

- 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

- *The Official GNOME 2 Developer's Guide*, by Matthias Warkus
- *Foundations of GTK+ Development*, by Andrew Krause
- 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
bill.lee@access-company.com