USING C++ TO IMPROVE PRODUCTIVITY IN PLATFORM WITH C API

CASE STUDY: NATIVE APP DEVELOPMENT IN TIZEN PLATFORM

ISSUES TO BE DISCUSSED

Several platform only provides Flat-C APIs to develop software on top of it. Using C to develop large application such as consumer apps or business apps takes longer time as C provides limited set of abstraction and reusable libraries.

On the other hand, C++ interfaces really smoothly with external libraries built using C or other procedural languages. In this poster, we want to bring up applicable use cases of C++ to develop a consumer application in a platform which only provides C APIs and how we integrated C API and C++ codes and patterns altogether.

BRIEFING TO TIZEN DEVELOPMENT

API REFERENCE

SDK DOWNLOAD

TFC SOURCE

SAMPLE

Tizen uses Enlightenment Framework (EFL) to build application on top of itself. EFL is a C-based API, similar with GTK library, which provides various infrastructure, from application loop, widget toolkit, and utility libraries. EFL API is structured with object-oriented paradigm but written completely in C-style. Therefore, many opaque pointers, `void*` data pointer, and repetitive function calls for setter and getter are expected in user codes.

Developing native application in Tizen use SDK packages consisting of Eclipse CDT-based IDE and native toolchain using either GCC or LLVM. These toolchains actually open up the possibilities of using C++ compiler to compile software targeting to Tizen platform. After further investigation, Tizen also includes C++ Standard Library binaries on its platform, therefore, enabling majorities of modern C++ features and STLs to be used.

TIZEN + C++ = TFC

We developed **Tizen Fundamental Classes (TFC)** as a framework library to bridge C++ classes with Tizen's EFL APIs. TFC provides design pattern and utilizes modern C++ features, thus provides an entirely new OOP environment for developer.

IMPLEMENTING MVC IN C++

There is no common libraries to develop MVC pattern in C++ for UI application, especially for EFL library. So, we designed TFC's MVC from scratch, following common rules of segregation of duties:

1. View handles the UI generation and behavior.
2. Controller handles the business logic.
3. Model defines the data to be processed by Controller and presented by View.

View and Controller classes have to inherit the base classes provided by TFC framework in order to be recognized by the framework components. As C++ does not (currently) provide reflection and dependency injection, TFC provides macros, template rules, as well as functions to easily integrate user-defined components with TFC internal components in less-verbose manner.

Code: ControllerManager implementation

```
void TFC::Framework::StackingControllerManager::PerformNavigation(
     char const* controllerName, ObjectClass* data,
     TFC::Framework::NavigationFlag mode) {

switch (mode) {
    case NavigationFlag::Back:
        return DoNavigateBackward();

    case NavigationFlag::ClearHistory:
        // Remove all controller from history
        while(!this->controllerStack.empty()) {
            this->CurrentController->Unload();
            PopView();
            PopController();
        }

        goto PerformNavigation_Default;

    case NavigationFlag::Default:
        PerformNavigation_Default:

        // Instantiate controller
        ControllerBase* newInstance = this->Instantiate(controllerName);

        // Perform OnLeave on previous controller
        if(!this->controllerStack.empty())
            this->CurrentController->Leave();

        PushController(newInstance);
        PushView(newInstance->View);

        // Instantiated State, move to Running state
        newInstance->Load(data);
        eventNavigationProcessed(this, newInstance);
        return;

    default:
        throw TFCException("Navigation mode not Implemented");
}
```

C++ MEMBER FUNCTION TO C CALLBACK

C has a limited type-safety feature. One common C idiom due to this limitation is passing `void*` pointer to refer to user data inside the callback. This require programmers to be very careful when passing and casting data between functions as this is prone to error.

Code: Creating View class and subscribing to event object

```
class SplashScreenView : public TFC::Framework::ViewBase {
private:
    Evas_Object* button { nullptr }; // EFL's opaque pointer
    EvasSmartEvent eventButtonClick; // Event Object
protected:
    // Will be called during the realization of View in EFL space
    virtual Evas_Object* CreateView(Evas_Object* root) override;
public:
    SplashScreenView();
    // Event callback
    void OnButtonClicked(Evas_Object* src, void* eventData);
};

SplashScreenView::SplashScreenView() {
    // Register instance function as function callback
    this->eventButtonClick += EventHandler(SomeView::OnButtonClicked);
}

SplashScreenView::CreateView(Evas_Object* root) {
    this->button = elm_button_add(root);
    evas_object_show(this->button);
    // Bind event object to EFL event
    this->eventButtonClick.Bind(this->button, "clicked");
    return this->button;
}
```

To solve this issue, TFC provides an object handler to wrap the type-casting processes in background, and enables programmer to use class member function to be called from C callbacks. Programmers does not required to perform casting and enable to use `this` keyword directly from the member function instead.

We got inspiration from C# programming style when implementing TFC's `EventObject` classes, where we overloaded the += operator to subscribe a member function to an event.

Code: Bind method implementation, calling EFL API

```
void EvasSmartEvent::Bind(Evas_Object* obj, const char* eventName) {
    // Call EFL API, pass our internal static function
    evas_object_smart_callback_add(
        obj, eventName, EvasSmartEvent::Callback, this);
}

// Static function in EvasSmartEvent Object
void EvasSmartEvent::Callback(void* data, Evas_Object* obj, void* info)  {
    // Cast back to EvasSmartEvent class, and call the function callback
    // registered in our event object
    (*reinterpret_cast<EvasSmartEvent*>(data))
            ->RaiseEvent(obj, reinterpret_cast<T>(info));
}
```

MEMORY MANAGEMENT

As C and C++ has different memory management model, we need to be careful when integrating C APIs with C++ codes and not introduce memory bug which can lead to security vulnerabilities. Constructor of C++ object can call C API which creates an instance of a C object. Therefore, during the destruction of the C++ object, the C object it previously acquired must be guaranteed to be released.

EFL library has its own memory management model with reference counting, which has to be taken into account when designing with C++ RAII (Resource Acquisition is Initialization) idiom. EFL widget is instantiated and managed within a tree data structure, which if the parent node is freed, every child of its own will be marked for deletion too, unless the reference is not yet reached zero due to explicitly acquiring its reference.

In TFC, EFL widget in a view is stored as "Naviframe" child, which acts like a stack. To remove a view, we "pop" the Naviframe, and the view will be cleaned by EFL. By maintaining the view class does not refer to any object other than what it is created inside the view class, we can guarantee that the code will access correct reference and avoid dangling reference issue.

CONCLUSION

Although Tizen development is not very common in general market, our TFC development can be a good example and learning how to develop native application in platform with limited set of APIs using C++ as the programming language. There are several key points that is required for C++ programmers to integrate Flat-C API library in their projects:

1. Investigate the necessary toolchain to compile C++ and the availability of C++ standard libraries in target platform.
2. Develop base codes with standardized software design pattern to handle basic integration between C APIs with C++ codes which has to be followed by the rest of C++ codes.
3. Eliminate unsafe typecasting behind base codes implementation, and only allow C++ codes to interact in type-safe manner.
4. Ensure no conflicting memory management between C++ and C codes. Especially when the C API supports a reference counting mechanism, ensure to release the reference on destructor.

OUR EXPERIENCES

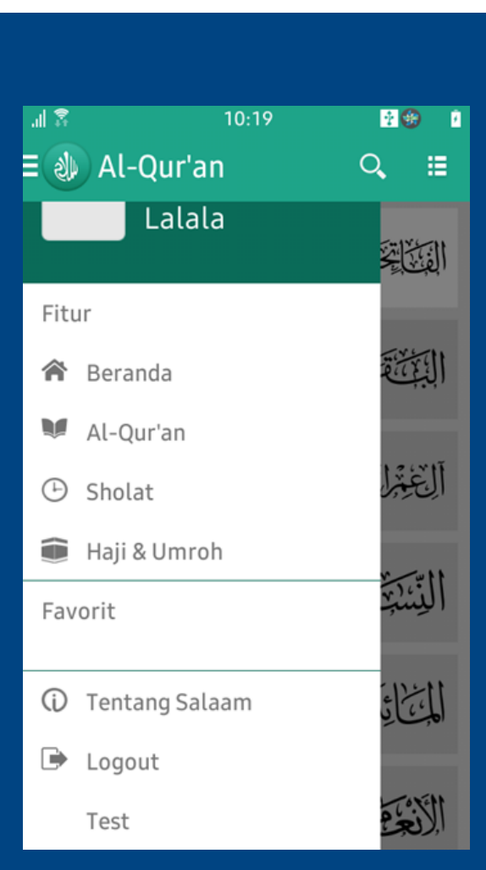
Our small team developed native Tizen App using C++ between 2015 and 2017. Most of our engineers has no experience in programming using C or C++, with some experiences in Android or Java programming. We managed to complete several apps, from local news apps, value-added preloaded apps, as well as Telegram Client for Tizen.

Jumat, 05/08/2016 16:16:40
Kapolri: Terduga Teroris di Batam Berencana Serang Singapura
Keenam terduga teroris yang ditangkap di Batam disebut berhubungan dengan Mujahidin Indonesia Timur. Mereka juga dituduh dekat dengan Bahrun Naim.

CNN Indonesia News App

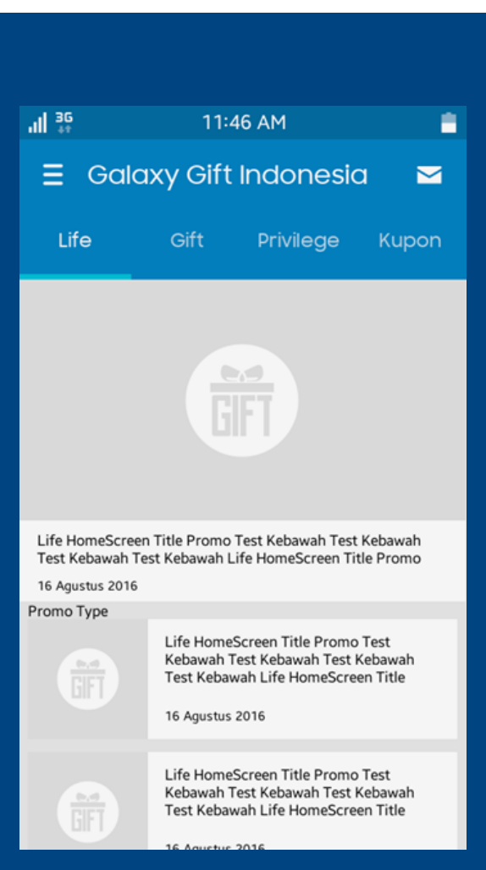
Teroris Asal Batam Berencana Teror Singapura bersama Bahrun Naim
Agus mengatakan, keduanya berencana menyerang Singapura dengan roket.
KPU DKI: Kita Tunggu Kejutan di Hari Akhir Penyerahan Syarat Calon Pe...

Kompas News App

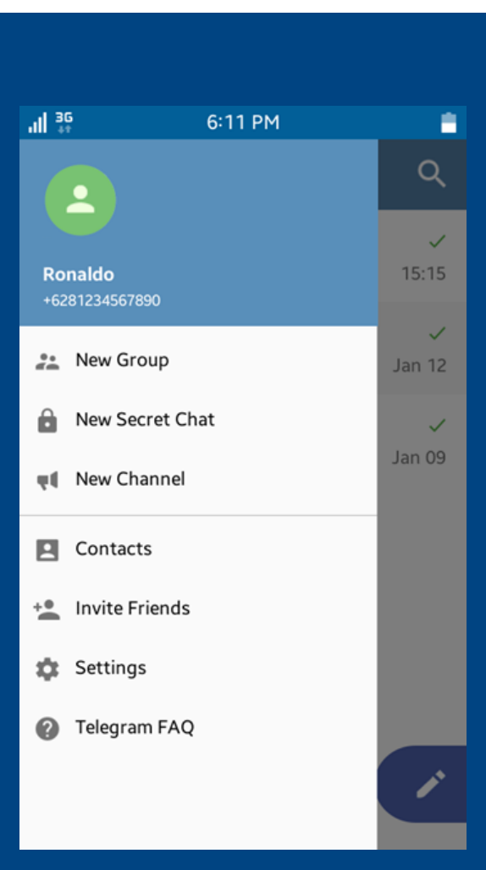


Salaam Islamic App

MyFaith Religious App



Samsung Gift Indonesia



Telegram for Tizen

TFC Introduction
Tizen Developer
Conference 2017

cppcon the c++ conference

ABSTRACT

Crafting Embedded Domain Specific Language (EDSL) in C++
September 24th at 14.00

TELEGRAM

PING Gilang!
via Telegram

Alexius Alvin
alexius_alvin@yahoo.com
Engineer at Samsung Research Indonesia (SRIN)

Gilang Mentari Hamidy
gilang.hamidy@gmail.com
Former Engineer at Samsung Research Indonesia (SRIN)

THE REST OF TIZEN TEAM OF SAMSUNG RESEARCH INDONESIA (SRIN)

Alif Pratama
Azzam Hanif
Bagas Prima Anugerah
Bany Pakha
Billy Austen
Calvin Windoro
Faidhon Nur
Ganda Utama
Hadi Prawiratama
Hansen
Hendrikus Juan Fernando
IB Putu Peradnya Dinata
Joey Gilian
Kevin Winata
Nizam Rahman
Puspa Anindita Yurianti
Pramono Winata
Rayhan Makarim
Risman Adnan
Yanuar Rahman