

Injected Bundle

- [Overview](#)
- [Implementation Details](#)
 - [How to Expose C/C++ API using injectedbundle](#)
 - [How to Invoke a JS API from C/C++](#)

Overview

Injected bundle is an Integration layer between Service Manager and the player in RDK Browser and WPE.

- Adds an ability to send messages from JavaScript to UI side and vice versa.
- Supports ACL for the JavaScript objects.

JavaScript Bridge implementation

The JS Bridge consists of 2 parts:

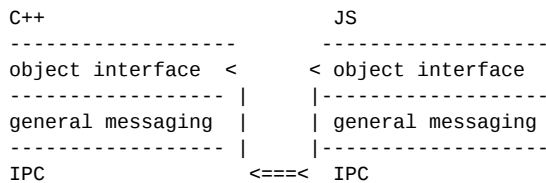
- execution backend (written in C++)
- client interface to execution backend (this one)

Each part consists of 2 layers:

- object interface (methods calls)
- general messaging (serialized messages)

Each method call JS => C++ is serialized into JSON, trasfered via IPC, received on another side, mapped into the corresponding method call.

So, the route is:



Responses and events are transferred in opposite direction (C++ => JS)

Implementation Details

How to Expose C/C++ API using injectedbundle

- Define an Injectedbundle CB that's invoked when a page is loaded

Define an object of WKBundlePageLoaderClientV1 with your own function callback for didCommitLoadForFrame. This callback gets invoked when a page's DOM contents finishes loading. This will also provide the mainFrame instance in which the JS object is to be loaded.

Example

```
WKBundlePageLoaderClientV1 client {
    {1, clientInfo},
    // Version 0.
    nullptr, // didStartProvisionalLoadForFrame;
    nullptr, // didReceiveServerRedirectForProvisionalLoadForFrame;
    nullptr, // didFailProvisionalLoadWithErrorForFrame;
    didCommitLoad, // didCommitLoadForFrame;
    nullptr, // didFinishDocumentLoadForFrame;
    nullptr, // didFinishLoadForFrame;
    nullptr, // didFailLoadWithErrorForFrame;
    nullptr, // didSameDocumentNavigationForFrame;
    nullptr, // didReceiveTitleForFrame;
    nullptr, // didFirstLayoutForFrame;
    nullptr, // didFirstVisuallyNonEmptyLayoutForFrame;
    nullptr, // didRemoveFrameFromHierarchy;
    nullptr, // didDisplayInsecureContentForFrame;
    nullptr, // didRunInsecureContentForFrame;
    nullptr, // didClearWindowObjectForFrame;
    nullptr, // didCancelClientRedirectForFrame;
    nullptr, // willPerformClientRedirectForFrame;
    nullptr, // didHandleOnloadEventsForFrame;

    // Version 1.
    nullptr, // didLayoutForFrame
    nullptr, // didNewFirstVisuallyNonEmptyLayout_unavailable
    nullptr, // didNewFirstVisuallyNonEmptyLayout_unavailable
    nullptr,
    nullptr, // globalObjectIsAvailableForFrame
    nullptr, // globalObjectIsAvailableForFrame
    nullptr, // didReconnectDOMWindowExtensionToGlobalObject
    nullptr // willDestroyGlobalObjectForDOMWindowExtension
};
```

- `void didCommitLoad(WKBundlePageRef page, WKBundleFrameRef frame)` // WKBundleFrameRef frame provides the JS context in which to load JS/C++ bindings

Example

```
void didCommitLoad(WKBundlePageRef page, WKBundleFrameRef frame)
{
    JSGlobalContextRef context = WKBundleFrameGetJavaScriptContext(frame);
    LoadJS(context);
}
```

- Using the context obtained from frame you can load custom JS objects, but first you need to define a JS class definition which provides a standard set of callbacks and properties for the JS object

Reference - `usr/include/JavaScriptCore/JSObjectRef.h`

```
typedef struct {
    int version; /* current (and only) version is 0 */
    JSCClassAttributes attributes;

    const char* className;
    JSCClassRef parentClass;

    const JSStaticValue* staticValues;
    const JSStaticFunction* staticFunctions;

    JSObjectInitializeCallback initialize;
    JSObjectFinalizeCallback finalize;
    JSObjectHasPropertyCallback hasProperty;
    JSObjectGetPropertyCallback getProperty;
    JSObjectSetPropertyCallback setProperty;
    JSObjectDeletePropertyCallback deleteProperty;
    JSObjectGetPropertyNamesCallback getPropertyNames;
    JSObjectCallAsFunctionCallback callAsFunction;
    JSObjectCallAsConstructorCallback callAsConstructor;
    JSObjectHasInstanceCallback hasInstance;
    JSObjectConvertToTypeCallback convertToType;
} JSCClassDefinition;
```

The important parameters for any JS object are `staticValues`, `staticFunctions` which helps to implement `JSObject.property` (get/set) and `JSObject.function()`

You may find more about the structure of `JSStaticValue` and `JSStaticFunction` in `JSObjectRef.h`

- Once the class and corresponding callbacks are defined, you may load this object onto the context that is retrieved from the frame object in `injectedbundle`

```
static const JSCClassDefinition JS_class_def;

void LoadJS(JSGlobalContextRef context)
{
    //Create a custom object that holds the context and any other variables that are required, for eg-
    session keeping var.
    struct CustomObject* obj = new CustomObject();
    JSCClassRef classDef = JSCClassCreate(&JS_class_def);
    JSObjectRef classObj = JSObjectMake(context, classDef, obj);
    JSObjectRef globalObj = JSContextGetGlobalObject(context);
    JSStringRef str = JSStringCreateWithUTF8CString("JSObject"); //If you would like to name your object as
    JSObject and access like JSObject.property
    JSObjectSetProperty(context, globalObj, str, classObj, kJSPPropertyAttributeReadOnly, NULL); //this
    loads the JSObject to page
    JSCClassRelease(classDef);
    JSStringRelease(str);
}
```

- Once the page is loaded, you may add an event listener for `onDOMContentLoaded` and verify if the `JSObject` is defined (`if(typeof global.JSObject !== "undefined" {console.log ("JSObject available")}`)

How to Invoke a JS API from C/C++

- Pass the JS API function pointer to Native code and store it as a `JSObjectRef`

```

JSObject.addEventListener(eventType, eventListener) //with eventListener having a definition like "function
eventListener(event: any) {...}"

static JSValueRef addEventListener(JSContextRef context, JSObjectRef function, JSObjectRef thisObject, size_t
argumentCount, const JSValueRef arguments[], JSValueRef *exception)
{
    JSObjectRef callbackObj = JSValueToObject(context, arguments[1], NULL);

    if (callbackObj != NULL && JSObjectIsFunction(context, callbackObj))
    {
        //Store the callbackObj in a variable
        eventListener = callbackObj;
        JSValueProtect(context, callbackObj);
    }
    else
    {
        return JSValueMakeUndefined(context);
    }

    return JSValueMakeNull(context);
}

```

- To invoke the JS API from C/C++, call `JSObjectCallAsFunction` with the `callbackObj` and arguments

```

JSObjectCallAsFunction(context, callbackObj, NULL, 1, args, NULL); //args is the list of arguments if
required

```