Allotropia plans, and past work around LOWA

allotropia

Collabora Online

LibreOffice Technology

Technical Day

COOL days

LibreOffice WebAssembly

- LibreOffice ported to WASM / WebAssembly
- Runs natively in all modern web browsers.
- Fully client-side.
- Built with the Emscripten toolchain
- Utilizes Qt as the UI

UNO in JavaScript with LOWA

- Having a direct interface on the browser enables wide range of applications.
  - e.g. custom native JavaScript UI for LOWA
  - Web apps that use embedded LOWA for various tasks.
  - e.g. LOWA could take care of (headless) extracting/exporting information from/to various documents.

Accessing C++ through JavaScript in the browser

- There were multiple possibilities for binding C++ to JavaScript:
  - Embind
  - WebIDL
  - Nbind
  - V8pp

    and possibly more...

UNO bindings with Embind

- Part of the emscripten toolchain
- Boost Python like bind semantics
- Relatively popular
- Raw & smart pointer support, automatic downcasting and more
  - Right now there's a very rough implementation in place. With lots of different bits unimplemented. Lots of room for improvement!

UNO bindings with Embind - Primary Bindings

```
// Reference bits
class_<BaseReference>("BaseReference");
enum_<UnoReference_Query>("UnoReference_Query").value("UNO_QUERY",
UNO_QUERY);

// Any
class_<Any>("Any").constructor(
    +[](const val& rObject, const TypeClass& rUnoType) -> Any {...}
);

class_<OUString>("OUString").constructor(...);
```

UNO bindings with Embind - Generating bindings from IDL

```
// com/sun/star/text/XTextRange.hdl

class SAL_NO_VTABLE SAL_DLLPUBLIC_RTTI XTextRange :
public ::css::uno::XInterface {

   // Methods
    virtual ::css::uno::Reference< ::css::text::XText > SAL_CALL getText() = 0;

}
```

```
class_<::css::text::XTextRange>("com$sun$star$text$XTextRange")
    // Bindings for methods
    .function("getText", &::css::text::XTextRange::getText)
    ;

class_<::css::uno::Reference<::css::text::XTextRange>,
            base<::css::uno::BaseReference>>("com$sun$star$text$XTextRangeRef")
    .constructor<::css::uno::BaseReference, ::css::uno::UnoReference_Query>()
    .function("getText", +[](::css::uno::Reference<::css::text::XTextRange>& self)
        { return self->getText(); }, allow_raw_pointers() )
    ;
```

UNO bindings with Embind - Primary Bindings

```
class_<Any>("uno_Any")
        .constructor(+[](const css::uno::Type& rUnoType, const val& rObject) -> Any {
            switch (rUnoType.getTypeClass())
            {
                case TypeClass_VOID:
                    return {};
                ...
                ...
                case TypeClass_SEQUENCE:
                case TypeClass_STRUCT:
                case TypeClass_EXCEPTION:
                case TypeClass_INTERFACE:
                {
                    emscripten::internal::EM_DESTRUCTORS destructors = nullptr;
                    emscripten::internal::EM_GENERIC_WIRE_TYPE result
                        = _emval_as(rObject.as_handle(), getTypeId(rUnoType), &destructors);
                    emscripten::internal::DestructorsRunner dr(destructors);
                    return css::uno::Any(
                        emscripten::internal::fromGenericWireType<void const*>(result), rUnoType);
                }
                case TypeClass_ENUM:
                {
                    emscripten::internal::EM_DESTRUCTORS destructors = nullptr;
                    emscripten::internal::EM_GENERIC_WIRE_TYPE result
                        = _emval_as(rObject.as_handle(), getTypeId(rUnoType), &destructors);
                    emscripten::internal::DestructorsRunner dr(destructors);
                    return css::uno::Any(
                        &o3tl::temporary(
                            emscripten::internal::fromGenericWireType<sal_Int32>(result)),
                        rUnoType);
                }
                default:
                    throw std::invalid_argument("bad type class");
            }
```

```
                case TypeClass_VOID:
                    return {};
                case TypeClass_BOOLEAN:
                    return Any{ rObject.as<bool>() };
                case TypeClass_BYTE:
                    return Any{ rObject.as<sal_Int8>() };
                case TypeClass_SHORT:
                    return Any{ rObject.as<sal_Int16>() };
                case TypeClass_UNSIGNED_SHORT:
                    return Any{ rObject.as<sal_uInt16>() };
                case TypeClass_LONG:
                    return Any{ rObject.as<sal_Int32>() };
                case TypeClass_UNSIGNED_LONG:
                    return Any{ rObject.as<sal_uInt32>() };
                case TypeClass_HYPER:
                    return Any{ rObject.as<sal_Int64>() };
                case TypeClass_UNSIGNED_HYPER:
                    return Any{ rObject.as<sal_uInt64>() };
                case TypeClass_FLOAT:
                    return Any{ rObject.as<float>() };
                case TypeClass_DOUBLE:
                    return Any{ rObject.as<double>() };
                case TypeClass_CHAR:
                    return Any{ rObject.as<char16_t>() };
                case TypeClass_STRING:
                    return
Any{ OUString(rObject.as<std::u16string>()) };
                case TypeClass_TYPE:
                    return
css::uno::Any(rObject.as<css::uno::Type>());
```

UNO bindings with Embind - Example

```
// inserts a string at the start of the Writer document.

let uno = init_unoembind_uno(Module);
let css = uno.com.sun.star;

xModel = Module.getCurrentModelFromViewSh();
xTextDocument = new css.text.XTextDocument(xModel.$query());
xText = xTextDocument.getText();
xSimpleText = new css.text.XSimpleText(xText.$query());
xTextCursor = xSimpleText.createTextCursor();
xTextRange = new css.text.XtextRange(xTextCursor.$query());

xTextRange.setString("string here!");

xModel.delete(); xTextDocument.delete(); xText.delete(); xSimpleText.delete();
xTextCursor.delete(); xTextRange.delete();
```

UNO bindings with Embind - Example

```
// changes each paragraph of the Writer document to a random color.

let uno = init_unoembind_uno(Module);
let css = uno.com.sun.star;
xModel = Module.getCurrentModelFromViewSh();
xEnumAccess = new css.container.XEnumerationAccess(xText.$query());
xParaEnumeration = xEnumAccess.createEnumeration();

while (xParaEnumeration.hasMoreElements()) {
    xParagraph = new css.text.XTextRange(xParaEnumeration.nextElement(),
Module.uno_Reference.FromAny);
    if (xParagraph.$is()) {
        xParaProps = new css.beans.XPropertySet(xParagraph.$query());
        let color = new Module.uno_Any(
            Module.uno_Type.Long(), Math.floor(Math.random() * 0xFFFFFF));
        xParaProps.setPropertyValue("CharColor", color);
        color.delete();
    }
}
```

UNO Exceptions

- We can already catch C++ exceptions from JavaScript.
- But the caught exception is just a pointer to the exception. With no apparent type attached…
- After the introduction of tags to WebAssembly. Embind introduced series of functions:
  - getCppExceptionTag
  - getCppExceptionThrownValue
  - getExceptionMessage

Probabaly with the new emscripten version (3.1.44) UNO Exception also can be handled.

Whats next?

- Make the API more JavaScript-ish
- Embinding more UNO Types
- Handling UNO Exceptions
- Creating/improving JavaScript binding bridge

Thank you for your attention

allotropia

LibreOffice Technology

Technical Day

COOL days