Forward compatibility of text_encoding with additional encoding registries

Revisions

| R1 | Rebase wording on P1885R9. Remove proposed normative guidance. Fully specify wording changes. Rework wording to decouple exposition-only text_encoding::id_ from the text_encoding::iana_id enumeration. |

Introduction

As currently proposed [1], text_encoding refers only to the Internet Assigned Numbers Authority (IANA) Character Sets database [2]. This registry is known to be incomplete and, in some respects, does not provide a perfect match to the requirements of C++ [2]. It is possible that future enhancements to text_encoding may wish to refer to additional/alternative registries.

Alter the names of text_encoding facilities that directly map IANA database data to explicitly reference iana.

Design

Do not rename aliases()

The text_encoding::aliases() member function currently returns a range of alternative names for a particular text_encoding. Although this range is required to include the aliases registered with IANA, it may also include additional, implementation-defined aliases.

This means that there is no need to rename this to iana_aliases(); the contract is already sufficiently wide to accommodate aliases from other registries.

Normative guidance for future compatibility

Currently, the exposition-only member variables of text_encoding contain only a text_encoding::id without scope for disambiguation of IDs or the capacity for representing non-IANA IDs, if required in the future.

This is adequate for now. Ideally we should provide normative guidance that implementors should consider the possibility of additional/alternative text encoding registries being used in the future and make accommodations in the layout of text_encoding, but anticipated theoretical future changes to an API are not implementable.

Instead, introduce an exposition only private enumeration type which represents an implementation-defined numeric identifier scheme which text_encoding::iana_mib() maps to text_encoding::iana_id.

As a slight specification cleanup, define almost all operations on a text_encoding in terms of calls to iana_mib(), with only iana_mib() making reference the exposition-only id_ member variable.
Proposed wording

Editing notes
All wording is relative to P1885R9 [4].

Update [text.encoding]:

```cpp
namespace std {
    struct text_encoding {

        inline constexpr size_t max_name_length = 63;

        enum class IanaId : int_least32_t {
            ...,
        };

        using enum id;

        constexpr text_encoding() noexcept = default;
        constexpr explicit text_encoding(string_view name) noexcept;
        constexpr text_encoding(IanaId id, mib) noexcept;

        constexpr IanaId IanaId(IanaMib() const noexcept;
        constexpr const char* name() const noexcept;

        struct aliases_view;
        constexpr aliases_view aliases() const noexcept;

        constexpr friend bool operator==(const text_encoding& encoding,
                                         const text_encoding & other) noexcept;
        constexpr friend bool operator==(const text_encoding& encoding,
                                         IanaId id, mib) noexcept;

        static constexpr text_encoding literal() noexcept;
        static text_encoding environment();
        template<IanaId id> static bool text_encoding::environment_is();

        private:
            enum id; // exposition only
            id IanaId; // exposition only
            char name_[max_name_length+1] = {0}; // exposition only
        };

    // hash support
    template<class T> struct hash;
    template<> struct hash<text_encoding>;
};
```

A registered character encoding is a character encoding scheme in the IANA Character Sets registry. 

[ Note: The IANA Character Sets registry refers to character sets rather than character encodings. — end note ]

The set of known registered character encoding contains every registered character encoding specified in the IANA Character Sets registry except for the following:
• NATS-DANO (33)
• NATS-DANO-ADD (34)

Each known registered character encoding is identified by an enumerator in `text_encoding:::id`, has a unique primary name and has a set of zero or more aliases. The primary name of a registered character encoding is the name of that encoding specified in the IANA Character Sets registry.

The set of aliases of a registered character encoding is an implementation-defined superset of the aliases specified in the IANA Character Sets registry. No two registered character encodings share any identical alias when compared by `COMP_NAME`.

[Note: The `text_encoding:::id` enumeration contains an enumerator for each known registered character encoding. For each encoding, the corresponding enumerator is derived from the alias beginning with “cs”, as follows:

- the “cs” prefix is removed from each name
- `csUnicode` is mapped to `text_encoding:::id::UCS2`
- `csIBM904` is mapped to `text_encoding:::id::IBM904`

— end note ]

How a `text_encoding` object is determined to be representative of a character encoding scheme implemented in the translation or execution environment is implementation-defined.

An object e of type text_encoding maintains the following invariants:

• e.name() == nullptr is true if and only if e.\texttt{\texttt{i}ana_mib()} == text_encoding:::id::unknown is true.
• e.\texttt{\texttt{i}ana_mib()} == text_encoding(e.name()).\texttt{\texttt{i}ana_mib()} is true if e.\texttt{\texttt{i}ana_mib()} == text_encoding:::\texttt{i}ana::id::other is true.

Recommended practice:

• Implementations should not consider registered encodings to be interchangeable [Example: Shift_JIS and Windows-31J denote different encodings].
• Implementations should not refer to a registered encoding to describe another similar yet different non-registered encoding unless there is a precedent on that implementation (Example: Big5).

Let bool \texttt{COMP_NAME}(string_view a, string_view b) be a function that returns true if the two strings a and b encoded in the ordinary literal encoding are equal ignoring, from left-to-right,

• all elements which are not digits or letters [character.seq.general],
• character case, and
• any sequence of one or more ‘0’ character not immediately preceded by a sequence consisting of a digit in the range [1-9] optionally followed by one or more elements which are not digits or letters.

[Note: This comparison is identical to the “Charset Alias Matching” algorithm described in the Unicode Technical Standard 22. — end note ]

[Example:}
assert(UTF_8, utf8) == true;
assert(UTF_8, utf8) == true;
assert(UTF_8, utf8) == false;
assert(UTF_8, utf8) == false;

— end example ]

costexpr text_encoding() noexcept;

Postconditions:

• iana_mib() == iana_id::unknown is true
• strlen(name_) == 0 is true

costexpr explicit text_encoding(string_view name) noexcept;

Preconditions:

• name represents a string in the ordinary literal encoding,
• all elements in name are in the basic source character set,
• name.size() <= max_name_length is true, and
• name.contains(\0) is false.

Postconditions:

• If there exists a primary name or alias a of a known registered character encoding such that COMP_NAME (a, name) is true, iana_mib() returns mib_ has the value of the enumerator of iana_id associated with that registered character encoding. Otherwise, iana_mib() mib_ == iana_id::other is true.
• name.compare(name_) == 0 is true

costexpr text_encoding(iana_id mib) noexcept;

Preconditions: mib has the value of one of the enumerators of iana_id.

Postconditions:

• iana_mib() mib_ == mib is true.
• If (iana_mib() mib_ == id::unknown || iana_mib() mib_ == id::other) is true, strlen(name_) == 0 is true. Otherwise, ranges::find(aliases, string_view(name_)) != aliases().end().

constexpr iana_id iana_mib() const noexcept;

Returns: The value of the enumerator of iana_id corresponding to id_mib_.

[ ... unchanged content omitted ... ]

template iana_id id
static bool text_encoding::environment_is();

Returns: environment() == id_

Update [text.encoding.comp]:

costexpr bool operator==(const text_encoding & a, const text_encoding & b)
noexcept;

Returns:
If \( \text{a.mib} == \text{b.mib} \) is true, then \( \text{COMP_NAME (a.name, b.name)} \).

Otherwise, \( \text{a.mib} == \text{b.mib} \).

Returns: encoding.

Remarks: This operator induces an equivalence relation on its arguments if and only if \( \text{mib} \) is true.

Acknowledgements
Thank you to Tom Honermann for suggesting making the IANA link explicit in identifiers and for proposing an improved wording strategy, and to Jens Maurer for highlighting the downsides of the way that IANA registry text encodings are specified.

References

