P2295 R4: Support for UTF-8 as portable source file extension

(D2295 R5 https://isocpp.org/files/papers/D2295R5.pdf)

Requirements identified by SG16 (meeting + e-mail review)

1. Wording based on P2314 R2 Character sets and encodings
2. UTF-8 source files shall be supported
3. Users shall be able to specify that source files are to be assumed to be UTF-8 encoded (n.b. outside the actual content of the file)
4. Files that were assumed to be UTF-8 encoded but contained some non-UTF-8 content shall be ill-formed
5. The contents of UTF-8 source files shall be transmitted to phase 2 of translation verbatim. There’s absolutely no implementation-defined behaviour allowed.

Figuring out a source file's encoding

[lex.phases.1]:

The encoding scheme of a physical source file is determined in an implementation defined manner. An implementation shall provide a means by which the encoding scheme of source files can be specified. [*Note:* A command-line option that specifies the encoding scheme to use ~~as the result of the determination~~ is such a means. — *end note*]

An implementation shall support the UTF-8 encoding scheme. The set of additional encodings supported by an implementation is implementation-defined.

Actually using the UTF-8 source code

[lex.phases.1]:

> If the encoding scheme of a physical source file is determined to be UTF-8, then the physical source file shall be a well-formed UTF-8 sequence. The source file is decoded to produce a sequence of UCS scalar values that constitutes the sequence of ~~representing~~ elements of the translation character set. [*Note:* There are no end-of-line indicators apart from the content of the UTF-8 sequence – *end note*]

[lex.charset], as modified by P2314 R2:

> 1. The translation character set consists of the following elements:
>   - each character named by ISO/IEC 10646, as identified by its unique UCS scalar value, and
>   - a distinct character for each UCS scalar value where no named character is assigned.

UTF-8 is an encoding scheme for all UCS scalar values. The translation character set is exactly the set of all UCS scalar values. No “mapping” occurs.

Everything else

[lex.phases.1]:

> For any other encoding scheme supported by the implementation, ~~P~~ physical source file characters are mapped, in an implementation-defined manner, to the translation character set (introducing new-line characters for end-of-line indicators). ~~The set of physical source file characters accepted is implementation-defined.~~

1. The encoding scheme of a physical source file is determined in an implementation defined manner. An implementation shall provide a mechanism to determine the encoding of a source file that is independent of its content. [Note: For example, an implementation can chose to provide a command line option to specify the expected encoding. — end note]
   An implementation shall support the UTF-8 encoding scheme. The set of additional encodings supported by an implementation is implementation-defined.
   If the encoding scheme of a physical source file is determined to be UTF-8, then the physical source file shall be a well-formed UTF-8 sequence representing elements of the translation character set.
   For any other encoding scheme supported by the implementation, ~~P~~physical source file characters are mapped, in an implementation-defined manner, to the translation character set (introducing new-line characters for end-of-line indicators). ~~The set of physical source file characters accepted is implementation-defined.~~
   An implementation may use any internal encoding, so long as an actual extended character encountered in the source file, and the same extended character expressed in the source file as a universal-character-name (e.g., using the \uXXXX notation), are handled equivalently except where this replacement is reverted in a raw string literal.
2. If the first character is U+FEFF BYTE ORDER MARK, it is deleted. Each instance of a backslash character (\) immediately followed by zero or more whitespace characters (other than new-line character) followed by a new-line character is deleted, splicing

P2362 R0: Make obfuscating wide character literals ill-formed

Wide non-encodable character literals L'🤦‍♀️'

- 32-bit wchar_t
  - There aren’t any non-encodable wide character literals
  - No change proposed to existing working code.
- 16-bit wchar_t
  - Wide character literals with codepoints outside the BMP are non-encodable
  - Wild implementation divergence:
    - Convert to UTF-16 and keep only a high surrogate, with diagnostic *off-by-default* (MSVC)
    - Convert to UTF-16 and keep only a low surrogate, with diagnostic *on-by-default* (GCC)
    - Error (clang)
  - Make non-encodable character literals ill-formed (clang behaviour).
  - Require implementations to enable diagnostics they already have.

Wide multicharacter literals   L'AB'

- A narrow character literal has type int. It can often fit multiple narrow characters.
- A wide character literal has type wchar_t. It can only fit exactly 1 wide character. Wide multicharacters always represent loss of information.
- Huge implementation divergence:
  - Throw away all but the *first* character, with diagnostic *off-by-default* (MSVC)
  - Throw away all but the *last* character, with diagnostic *on-by-default* (GCC and clang)
- Horrible miscompilation of e.g. decomposed L'é' ensues
- Make wide multicharacter literals ill-formed
- Require implementations to enable diagnostics they already have.