Unicode and UTF-8 Conversion Functions in C++

The `IceUtil` namespace contains two helper functions that allow you to convert between wide strings containing Unicode characters (either 16- or 32-bit, depending on your native `wchar_t` size) and narrow strings in UTF-8 encoding:

C++

```
enum ConversionFlags { strictConversion, lenientConversion };

std::string wstringToString(const std::wstring&, ConversionFlags = lenientConversion);
std::wstring stringToWstring(const std::string&, ConversionFlags = lenientConversion);
```

These functions always convert to and from UTF-8 encoding, that is, they ignore any locale setting that might specify a different encoding.

Byte sequences that are illegal, such as `0xF4908080`, result in a `UTFConversionException`. For other errors, the `ConversionFlags` parameter determines how rigorously the functions check for errors. When set to `lenientConversion` (the default), the functions tolerate isolated surrogates and irregular sequences, and substitute the UTF-32 replacement character `0x0000FFFD` for character values above `0x10FFFF`. When set to `strictConversion`, the functions do not tolerate such errors and throw a `UTFConversionException` instead:

C++

```
enum ConversionError { partialCharacter, badEncoding };

class UTFConversionException : public Exception {
public:
    UTFConversionException(const char* file, int line, ConversionError r);

    ConversionError conversionError() const;
    // ...
};
```

The `conversionError` member function returns the reason for the failure:

- `partialCharacter`
  The UTF-8 source string contains a trailing incomplete UTF-8 byte sequence.
- `badEncoding`
  The UTF-8 source string contains a byte sequence that is not a valid UTF-8 encoded character, or the Unicode source string contains a bit pattern that does not represent a valid Unicode character.