Opinionated notes on the Windows.Data.Json namespace

devblogs.microsoft.com/oldnewthing/20230102-00

January 2, 2023

Raymond Chen

The Windows Runtime provides types in the `Windows.Data.Json` namespace for dealing with JSON. You can parse a JSON string into a JSON object model, and you can conversely build a JSON object model and then convert it back to a string.

Here are some of my opinionated notes on these classes.

In order to be language-independent, the objects in the Windows.Data.Json namespace are COM objects, rather than native objects in C++, C#, or whatever your language is. This means that calls to the methods go through vtable dispatch, which the compiler cannot optimize. This is a fundamental limitation of using a language-independent object model: You cannot take advantage of language-specific optimizations.

There are a lot of interfaces, and calling a method on an interface different from the one you have in your hand requires you do perform a `QueryInterface` to switch to the interface that has the method.

```
// C++/WinRT
JsonObject jsonObject = ...;
jsonObject.Insert(key, value);
```

Insert is a method on `IMap<String, IJsonValue>`, so under the covers, there is an interface query.

```
IMap<hstring, IJsonValue> map;
jsonObject.QueryInterface(IID_PPV_ARGS(map.put()));
map.Insert(key, value);
// map calls Release() on destruction
```

The above code could have avoided two COM calls (the `QueryInterface` and the `Release`) by using the bespoke `SetNamedValue` method.

```
// C++/WinRT
JsonObject jsonObject = ...;
jsonObject.SetNamedValue(key, value);
```

The `SetNamedValue` method is a method on `IJsonObject`, so you can call it without having to change interfaces.

The `JsonObject` is awkward to use in an exception-free manner. In order to read a value if it is present, most people write something like

```
if (jsonObject.HasKey(L"name")) {
    auto value = jsonObject.GetNamedValue(L"name");
    if (value.ValueType() == JsonValueType::String) {
        auto name = value.GetString();
    }
}
```

This is a double-query, which is made even more expensive when you realize that `HasKey` is a method on `IMap`, so you also have an interface query/release hiding in there.

There's a little-known overload of `GetNamedValue` that lets you specify what to return if the value isn't found:

```
auto value = jsonObject.GetNamedValue(L"name", nullptr);
if (value && value.ValueType() == JsonValueType::String) {
    auto name = value.GetString();
}
```

Since a present named value never has a `nullptr` value, we can be confident that if `nullptr` is returned, then it means that the value was not present. (If the associated value is a JSON `null`, it is returned as a non-null object whose value type is `JsonValueType::Null`.)

There's a corresponding little-known overload of `GetNamedString` that returns the string, or a fallback value if the string is not present.

```
auto name = jsonObject.GetNamedString(L"name", L"untitled");
```

Choosing a fallback value for a missing string is trickier because there is no out-of-band value that unmistakably indicates that the fallback was returned. (Recall that a `nullptr` `HSTRING` represents the empty string.) The `GetNamedBoolean` function suffers from the same problem. For `GetNamedValue` and `GetNamedArray`, you can use `nullptr`, and for `GetNamedNumber` you can use NaN or one of the Infinity values, since those are not legal in JSON.

There's still a hidden trap in the `GetNamed...` functions with fallback: If the value is present but is not the type you expect, then instead of returning the fallback, you get an `E_ILLEGAL_METHOD_CALL` error, which usually projects as an exception. This is a problem if you're trying to be exception-free yet resilient to JSON that doesn't match your schema. I think the best you can do is `GetNamedValue` and then check the `ValueType` before converting.

The JSON parsing and serialization methods are not configurable. Although there is a JSON specification, there is wide disagreement over what is legal JSON when you get to the edges of the specification. I've put a conformance report at the end of this article.

One thing that stands out from the conformance report is that the `TryParse` method can throw an exception if the JSON string is legal but not representable as a `JsonValue` object, or if an implementation limit is reached before the string can be fully validated. So even though you think you're avoiding exceptions by using `JsonValue::TryParse`, you aren't actually exception-free.

Anyway, back to lack of configurability: Since you cannot configure the input, you cannot specify which variant of JSON you want to accept. And since you cannot configure the output, you cannot ask for pretty-printing. This makes the `Windows.Data.Json` objects unsuitable for generating JSON configuration files which are intended to be human-edited.

Note also that the `Windows.Data.Json` interconversion functions consume and produce UTF-16LE strings. Most of the time, the original JSON data in UTF-8 format, and the final output is also in UTF-8 format, so you have extra conversion steps on either side. Of course, this isn't a problem if your I/O functions already do that conversion for you. For example, if you ask `HttpClient` for the string content, it returns the string in UTF16-LE format, ready to be handed to `JsonValue::TryParse`.

With all of these caveats, it sure sounds like the `Windows.Data.Json` namespace is terrible. Why would you ever want to use it?

Well, it's already there.

If you already require Windows 8 or higher, then these classes are already present, and you can consume them without having to add another dependency to your project. This is important if you are concerned about disk footprint or download size, or just want to minimize your dependencies. For example, I have a few internal tools in which the program itself is 60KB, but the dependencies to do the Web authentication are 300KB.

Also, if parsing JSON is not a performance-critical operation in your program, you may figure that the inefficiencies of a language-independent library (compared to a native-language library) aren't really a big deal. For example, if your program is parsing moderate-sized JSON received from a Web server, any time savings by switching to a highly-optimized JSON parser is almost certainly going to be overwhelmed by the network I/O.[1]

Bonus chatter: The classes in the `Windows.Data.Json` namespace are provided by the Windows Runtime as a convenience. No other parts of the API surface require it.[2] Any methods that accept JSON do so in the form of a string, so you are welcome to use whatever JSON library you like.

Appendix: Here's the JSON conformance report, generated from Nicolas Seriot's JSON test suite.

Test	Result	Notes
i_number_double_huge_neg_exp	Exception `WEB_E_INVALID_JSON_NUMBER`	
i_number_huge_exp	Exception `WEB_E_INVALID_JSON_NUMBER`	
i_number_neg_int_huge_exp	Exception `WEB_E_INVALID_JSON_NUMBER`	
i_number_pos_double_huge_exp	Exception `WEB_E_INVALID_JSON_NUMBER`	
i_number_real_neg_overflow	Exception `WEB_E_INVALID_JSON_NUMBER`	
i_number_real_pos_overflow	Exception `WEB_E_INVALID_JSON_NUMBER`	
i_number_real_underflow	Exception `WEB_E_INVALID_JSON_NUMBER`	
i_number_too_big_neg_int	Accept	Allowed
i_number_too_big_pos_int	Accept	Allowed
i_number_very_big_negative_int	Accept	Allowed
i_object_key_lone_2nd_surrogate	Accept	Allowed
i_string_1st_surrogate_but_2nd_missing	Accept	Allowed
i_string_1st_valid_surrogate_2nd_invalid	Accept	Allowed
i_string_incomplete_surrogates_escape_valid	Accept	Allowed
i_string_incomplete_surrogate_and_escape_valid	Accept	Allowed
i_string_incomplete_surrogate_pair	Accept	Allowed
i_string_invalid_lonely_surrogate	Accept	Allowed
i_string_invalid_surrogate	Accept	Allowed
i_string_invalid_utf-8	Accept	Allowed
i_string_inverted_surrogates_U+1D11E	Accept	Allowed

i_string_iso_latin_1	Accept	Allowed
i_string_lone_second_surrogate	Accept	Allowed
i_string_lone_utf8_continuation_byte	Accept	Allowed
i_string_not_in_unicode_range	Accept	Allowed
i_string_overlong_sequence_2_bytes	Accept	Allowed
i_string_overlong_sequence_6_bytes	Accept	Allowed
i_string_overlong_sequence_6_bytes_null	Accept	Allowed
i_string_truncated-utf-8	Accept	Allowed
i_string_UTF-16LE_with_BOM	Accept	Allowed
i_string_UTF-8_invalid_sequence	Accept	Allowed
i_string_utf16BE_no_BOM	Reject	Allowed
i_string_utf16LE_no_BOM	Reject	Allowed
i_string_UTF8_surrogate_U+D800	Accept	Allowed
i_structure_500_nested_arrays	Accept	Allowed
i_structure_UTF-8_BOM_empty_object	Accept	Allowed
n_array_1_true_without_comma	Reject	OK
n_array_a_invalid_utf8	Reject	OK
n_array_colon_instead_of_comma	Reject	OK
n_array_comma_after_close	Reject	OK
n_array_comma_and_number	Reject	OK
n_array_double_comma	Reject	OK
n_array_double_extra_comma	Reject	OK
n_array_extra_close	Reject	OK
n_array_extra_comma	Reject	OK
n_array_incomplete	Reject	OK
n_array_incomplete_invalid_value	Reject	OK

n_array_inner_array_no_comma	Reject	OK
n_array_invalid_utf8	Reject	OK
n_array_items_separated_by_semicolon	Reject	OK
n_array_just_comma	Reject	OK
n_array_just_minus	Reject	OK
n_array_missing_value	Reject	OK
n_array_newlines_unclosed	Reject	OK
n_array_number_and_comma	Reject	OK
n_array_number_and_several_commas	Reject	OK
n_array_spaces_vertical_tab_formfeed	Reject	OK
n_array_star_inside	Reject	OK
n_array_unclosed	Reject	OK
n_array_unclosed_trailing_comma	Reject	OK
n_array_unclosed_with_new_lines	Reject	OK
n_array_unclosed_with_object_inside	Reject	OK
n_incomplete_false	Reject	OK
n_incomplete_null	Reject	OK
n_incomplete_true	Reject	OK
n_multidigit_number_then_00	Reject	OK
n_number_++	Reject	OK
n_number_+1	Reject	OK
n_number_+Inf	Reject	OK
n_number_-01	Reject	OK
n_number_-1.0.	Reject	OK
n_number_-2.	Reject	OK
n_number_-NaN	Reject	OK

n_number_.-1	Reject	OK
n_number_.2e-3	Reject	OK
n_number_0.1.2	Reject	OK
n_number_0.3e+	Reject	OK
n_number_0.3e	Reject	OK
n_number_0.e1	Reject	OK
n_number_0e+	Reject	OK
n_number_0e	Reject	OK
n_number_0_capital_E+	Reject	OK
n_number_0_capital_E	Reject	OK
n_number_1.0e+	Reject	OK
n_number_1.0e-	Reject	OK
n_number_1.0e	Reject	OK
n_number_1eE2	Reject	OK
n_number_1_000	Reject	OK
n_number_2.e+3	Reject	OK
n_number_2.e-3	Reject	OK
n_number_2.e3	Reject	OK
n_number_9.e+	Reject	OK
n_number_expression	Reject	OK
n_number_hex_1_digit	Reject	OK
n_number_hex_2_digits	Reject	OK
n_number_Inf	Reject	OK
n_number_infinity	Reject	OK
n_number_invalid+-	Reject	OK
n_number_invalid-negative-real	Reject	OK

n_number_invalid-utf-8-in-bigger-int	Reject	OK
n_number_invalid-utf-8-in-exponent	Reject	OK
n_number_invalid-utf-8-in-int	Reject	OK
n_number_minus_infinity	Reject	OK
n_number_minus_sign_with_trailing_garbage	Reject	OK
n_number_minus_space_1	Reject	OK
n_number_NaN	Reject	OK
n_number_neg_int_starting_with_zero	Reject	OK
n_number_neg_real_without_int_part	Reject	OK
n_number_neg_with_garbage_at_end	Reject	OK
n_number_real_garbage_after_e	Reject	OK
n_number_real_without_fractional_part	Reject	OK
n_number_real_with_invalid_utf8_after_e	Reject	OK
n_number_starting_with_dot	Reject	OK
n_number_U+FF11_fullwidth_digit_one	Reject	OK
n_number_with_alpha	Reject	OK
n_number_with_alpha_char	Reject	OK
n_number_with_leading_zero	Reject	OK
n_object_bad_value	Reject	OK
n_object_bracket_key	Reject	OK
n_object_comma_instead_of_colon	Reject	OK
n_object_double_colon	Reject	OK
n_object_emoji	Reject	OK
n_object_garbage_at_end	Reject	OK
n_object_key_with_single_quotes	Reject	OK
n_object_lone_continuation_byte_in_ key_and_trailing_comma	Reject	OK

n_object_missing_colon	Reject	OK
n_object_missing_key	Reject	OK
n_object_missing_semicolon	Reject	OK
n_object_missing_value	Reject	OK
n_object_no-colon	Reject	OK
n_object_non_string_key	Reject	OK
n_object_non_string_key_ but_huge_number_instead	Reject	OK
n_object_repeated_null_null	Reject	OK
n_object_several_trailing_commas	Reject	OK
n_object_single_quote	Reject	OK
n_object_trailing_comma	Reject	OK
n_object_trailing_comment	Reject	OK
n_object_trailing_comment_open	Reject	OK
n_object_trailing_comment_slash_open	Reject	OK
n_object_trailing_comment_slash_open_incomplete	Reject	OK
n_object_two_commas_in_a_row	Reject	OK
n_object_unquoted_key	Reject	OK
n_object_unterminated-value	Reject	OK
n_object_with_single_string	Reject	OK
n_object_with_trailing_garbage	Reject	OK
n_single_space	Reject	OK
n_string_1_surrogate_then_escape	Reject	OK
n_string_1_surrogate_then_escape_u	Reject	OK
n_string_1_surrogate_then_escape_u1	Reject	OK
n_string_1_surrogate_then_escape_u1x	Reject	OK
n_string_accentuated_char_no_quotes	Reject	OK

n_string_backslash_00	Reject	OK
n_string_escaped_backslash_bad	Reject	OK
n_string_escaped_ctrl_char_tab	Reject	OK
n_string_escaped_emoji	Reject	OK
n_string_escape_x	Reject	OK
n_string_incomplete_escape	Reject	OK
n_string_incomplete_escaped_character	Reject	OK
n_string_incomplete_surrogate	Reject	OK
n_string_incomplete_surrogate_escape_invalid	Reject	OK
n_string_invalid-utf-8-in-escape	Reject	OK
n_string_invalid_backslash_esc	Reject	OK
n_string_invalid_unicode_escape	Reject	OK
n_string_invalid_utf8_after_escape	Reject	OK
n_string_leading_uescaped_thinspace	Reject	OK
n_string_no_quotes_with_bad_escape	Reject	OK
n_string_single_doublequote	Reject	OK
n_string_single_quote	Reject	OK
n_string_single_string_no_double_quotes	Reject	OK
n_string_start_escape_unclosed	Reject	OK
n_string_unescaped_ctrl_char	Reject	OK
n_string_unescaped_newline	Reject	OK
n_string_unescaped_tab	Reject	OK
n_string_unicode_CapitalU	Reject	OK
n_string_with_trailing_garbage	Reject	OK
n_structure_100000_opening_arrays	Exception `ERROR_IMPLEMENTATION_LIMIT`	
n_structure_angle_bracket_.	Reject	OK

n_structure_angle_bracket_null	Reject	OK
n_structure_array_trailing_garbage	Reject	OK
n_structure_array_with_extra_array_close	Reject	OK
n_structure_array_with_unclosed_string	Reject	OK
n_structure_ascii-unicode-identifier	Reject	OK
n_structure_capitalized_True	Reject	OK
n_structure_close_unopened_array	Reject	OK
n_structure_comma_instead_of_closing_brace	Reject	OK
n_structure_double_array	Reject	OK
n_structure_end_array	Reject	OK
n_structure_incomplete_UTF8_BOM	Reject	OK
n_structure_lone-invalid-utf-8	Reject	OK
n_structure_lone-open-bracket	Reject	OK
n_structure_no_data	Reject	OK
n_structure_null-byte-outside-string	Reject	OK
n_structure_number_with_trailing_garbage	Reject	OK
n_structure_object_followed_by_closing_object	Reject	OK
n_structure_object_unclosed_no_value	Reject	OK
n_structure_object_with_comment	Reject	OK
n_structure_object_with_trailing_garbage	Reject	OK
n_structure_open_array_apostrophe	Reject	OK
n_structure_open_array_comma	Reject	OK
n_structure_open_array_object	Exception `ERROR_IMPLEMENTATION_LIMIT`	
n_structure_open_array_open_object	Reject	OK
n_structure_open_array_open_string	Reject	OK
n_structure_open_array_string	Reject	OK

n_structure_open_object	Reject	OK
n_structure_open_object_close_array	Reject	OK
n_structure_open_object_comma	Reject	OK
n_structure_open_object_open_array	Reject	OK
n_structure_open_object_open_string	Reject	OK
n_structure_open_object_string_with_apostrophes	Reject	OK
n_structure_open_open	Reject	OK
n_structure_single_eacute	Reject	OK
n_structure_single_star	Reject	OK
n_structure_trailing_#	Reject	OK
n_structure_U+2060_word_joined	Reject	OK
n_structure_uescaped_LF_before_string	Reject	OK
n_structure_unclosed_array	Reject	OK
n_structure_unclosed_array_partial_null	Reject	OK
n_structure_unclosed_array_unfinished_false	Reject	OK
n_structure_unclosed_array_unfinished_true	Reject	OK
n_structure_unclosed_object	Reject	OK
n_structure_unicode-identifier	Reject	OK
n_structure_UTF8_BOM_no_data	Reject	OK
n_structure_whitespace_formfeed	Reject	OK
n_structure_whitespace_U+2060_word_joiner	Reject	OK
y_array_arraysWithSpaces	Accept	OK
y_array_empty-string	Accept	OK
y_array_empty	Accept	OK
y_array_ending_with_newline	Accept	OK
y_array_false	Accept	OK

y_array_heterogeneous	Accept	OK
y_array_null	Accept	OK
y_array_with_1_and_newline	Accept	OK
y_array_with_leading_space	Accept	OK
y_array_with_several_null	Accept	OK
y_array_with_trailing_space	Accept	OK
y_number	Accept	OK
y_number_0e+1	Accept	OK
y_number_0e1	Accept	OK
y_number_after_space	Accept	OK
y_number_double_close_to_zero	Accept	OK
y_number_int_with_exp	Accept	OK
y_number_minus_zero	Accept	OK
y_number_negative_int	Accept	OK
y_number_negative_one	Accept	OK
y_number_negative_zero	Accept	OK
y_number_real_capital_e	Accept	OK
y_number_real_capital_e_neg_exp	Accept	OK
y_number_real_capital_e_pos_exp	Accept	OK
y_number_real_exponent	Accept	OK
y_number_real_fraction_exponent	Accept	OK
y_number_real_neg_exp	Accept	OK
y_number_real_pos_exponent	Accept	OK
y_number_simple_int	Accept	OK
y_number_simple_real	Accept	OK
y_object	Accept	OK

y_object_basic	Accept	OK
y_object_duplicated_key	Accept	OK
y_object_duplicated_key_and_value	Accept	OK
y_object_empty	Accept	OK
y_object_empty_key	Accept	OK
y_object_escaped_null_in_key	Accept	OK
y_object_extreme_numbers	Accept	OK
y_object_long_strings	Accept	OK
y_object_simple	Accept	OK
y_object_string_unicode	Accept	OK
y_object_with_newlines	Accept	OK
y_string_1_2_3_bytes_UTF-8_sequences	Accept	OK
y_string_accepted_surrogate_pair	Accept	OK
y_string_accepted_surrogate_pairs	Accept	OK
y_string_allowed_escapes	Accept	OK
y_string_backslash_and_u_escaped_zero	Accept	OK
y_string_backslash_doublequotes	Accept	OK
y_string_comments	Accept	OK
y_string_double_escape_a	Accept	OK
y_string_double_escape_n	Accept	OK
y_string_escaped_control_character	Accept	OK
y_string_escaped_noncharacter	Accept	OK
y_string_in_array	Accept	OK
y_string_in_array_with_leading_space	Accept	OK
y_string_last_surrogates_1_and_2	Accept	OK
y_string_nbsp_uescaped	Accept	OK

y_string_nonCharacterInUTF-8_U+10FFFF	Accept	OK
y_string_nonCharacterInUTF-8_U+FFFF	Accept	OK
y_string_null_escape	Accept	OK
y_string_one-byte-utf-8	Accept	OK
y_string_pi	Accept	OK
y_string_reservedCharacterInUTF-8_U+1BFFF	Accept	OK
y_string_simple_ascii	Accept	OK
y_string_space	Accept	OK
y_string_surrogates_U+1D11E_ MUSICAL_SYMBOL_G_CLEF	Accept	OK
y_string_three-byte-utf-8	Accept	OK
y_string_two-byte-utf-8	Accept	OK
y_string_u+2028_line_sep	Accept	OK
y_string_u+2029_par_sep	Accept	OK
y_string_uEscape	Accept	OK
y_string_uescaped_newline	Accept	OK
y_string_unescaped_char_delete	Accept	OK
y_string_unicode	Accept	OK
y_string_unicodeEscapedBackslash	Accept	OK
y_string_unicode_2	Accept	OK
y_string_unicode_escaped_double_quote	Accept	OK
y_string_unicode_U+10FFFE_nonchar	Accept	OK
y_string_unicode_U+1FFFE_nonchar	Accept	OK
y_string_unicode_U+200B_ZERO_WIDTH_SPACE	Accept	OK
y_string_unicode_U+2064_invisible_plus	Accept	OK
y_string_unicode_U+FDD0_nonchar	Accept	OK
y_string_unicode_U+FFFE_nonchar	Accept	OK

y_string_utf8	Accept	OK
y_string_with_del_character	Accept	OK
y_structure_lonely_false	Accept	OK
y_structure_lonely_int	Accept	OK
y_structure_lonely_negative_real	Accept	OK
y_structure_lonely_null	Accept	OK
y_structure_lonely_string	Accept	OK
y_structure_lonely_true	Accept	OK
y_structure_string_empty	Accept	OK
y_structure_trailing_newline	Accept	OK
y_structure_true_in_array	Accept	OK
y_structure_whitespace_array	Accept	OK
number_-9223372036854775808	`[-9.2233720368547758E+18]`	
number_-9223372036854775809	`[-9.2233720368547758E+18]`	
number_1.0	`[1]`	
number_1.000000000000000005	`[1]`	
number_1000000000000000	`[1E+15]`	
number_10000000000000000999	`[1E+19]`	
number_1e-999	Exception `WEB_E_INVALID_JSON_NUMBER`	
number_1e6	`[1000000]`	
number_9223372036854775807	`[9.2233720368547758E+18]`	
number_9223372036854775808	[9.2233720368547758E+18]	
object_key_nfc_nfd	`{"C3A9":"NFC","65CC81":"NFD"}`	
object_key_nfd_nfc	`{"65CC81":"NFD","C3A9":"NFC"}`	
object_same_key_different_values	`{"a":2}`	
object_same_key_same_value	`{"a":1}`	

object_same_key_unclear_values	`{"a":-0}`
string_1_escaped_invalid_codepoint	`["EFBFBD"]`
string_1_invalid_codepoint	N/A
string_2_escaped_invalid_codepoints	`["EFBFBDEFBFBD"]`
string_2_invalid_codepoints	N/A
string_3_escaped_invalid_codepoints	`["EFBFBDEFBFBDEFBFBD"]`
string_3_invalid_codepoints	N/A
string_with_escaped_NULL	`["A\u0000B"]`

For all of the “invalid codepoint” tests, the EFBFBD sequence is an encoded U+FFFD REPLACEMENT CHARACTER.

The “raw invalid codepoint” tests are marked N/A because the failure is in the conversion from UTF-8 to UTF-16LE, which is something the caller does before calling `JsonValue::TryParse`.

1 Though not always.

2 The `Windows.System.Diagnostics.DiagnosticInvoker.RunDiagnosticActionAsync` method does require that you use a `Windows.Data.Json.JsonObject`, but this was a mistake, which was corrected by the addition of the `Windows.System.Diagnostics.DiagnosticInvoker.RunDiagnosticActionFromStringAsync` method, which accepts a plain string. You can generate that string using whatever JSON library you choose.

Raymond Chen

Follow