JSON Implementation

XINA API actions and responses are encoded in JSON, a common, widely supported, human-readable format. For the most part, the implementation of JSON in XINA is intended to match the format specification as closesly as possible. In the interest of increased flexibility and greater clarity, the XINA JSON parser does support some extensions and enforce additional conditions which are not part of the standard. However, **all XINA API JSON outputs will always be valid standard JSON**.

Empty Strings

XINA treats an empty string as equal to the JSON literal `null`. For example, the following two objects are equal in XINA:

```
{ "foo": "" }
```

```
{ "foo": null }
```

This is based on the XINA interpretation of `null` as meaning "no value", and an empty string also effectively representing "no value". Therefore, XINA does not permit an empty string as an object key, because `null` would not be a valid object key. For example, this is valid JSON, but not valid in XINA:

```
{ "": "foo" }
```

Duplicate Keys

The JSON standard permits (but discourages) duplicate keys in an object. XINA does not permit this, as the interpretation would be ambiguous. For example, this is valid JSON but not valid in XINA:

```
{
 "foo": "bar",
 "foo": "not bar?!"
}
```

Case Insensitive Keys

The JSON standard treats object keys as case sensitive, but XINA treats them as case-insensitive, for consistency with the overall case-insentitive design of the API, and to reduce abiguity. For example, this object would cause an error, due the keys being duplicates:

```
{
 "foo": true,
 "FOO": false
}
```

Normalized Keys

XINA normalizes object keys, meaning any leading and trailing white space is trimmed, and any internal white space is reduced to a single space. Again, this is to reduce ambiguity. For example, this object would cause an error, due the keys being duplicates:

```
{
 "foo bar": true,
 " foo  bar ": false
}
```

Extra Commas

XINA permits extra commas after the last item in arrays and objects. This does not affect the interpretation of the data.

```
[
 "foo",
 "bar",
]
```

```
{
 "foo": true,
 "bar": false,
}
```

Numeric Parsing

The JSON standard does not explicitly define rules around numeric parsing, aside from the text format for numbers. XINA initially parses numeric values with (virtually) unlimited precision. If the value is a whole number, it must fit in the range of a signed 64-bit integer (greater or equal to -2^{63} and less or equal to $2^{63}-1$). This applies even if the value has a fraction component, so long as it is zero. Otherwise it must fit into range representable by a 64-bit floating point value, $\pm(2-2^{-52})\cdot 2^{1023}$.

In summary:

- whole numbers must be precisely representable by a 64-bit signed integer

- decimal number magnitude must be accurately representable by a 64-bit floating point, but precision may be lost

Explicit Undefined

JSON does not include the JavaScript `undefined` literal, but XINA will parse it. For example, the following two objects are parsable and equivalent in XINA:

```
{
 "foo": true,
 "bar": undefined
}
```

```
{
 "foo": true
}
```

In practice this is rarely needed. In most cases a `null` literal will essentially serve this role, except in XINA record objects, where an explicit `null` refers to a `null` field value, and `undefined` refers to nothing.

Revision #11
Created 15 January 2026 20:43:13 by Nick Dobson
Updated 21 January 2026 19:43:20 by Nick Dobson