XDelta Compression Format

Overview

The XINA Delta (xd) compression format is a simple, high performance format for compression and decompression of a list of numeric values into a string. The compression is most efficient for data sets with frequently repeated values, or predictable value iterations.

The format consists of a few main features:

- a comma separated list of plain text values
- `blank` values indicating either repeating the previous value, or a static value
- `"XnY"` indicates repeating value `X` exactly `Y` times
- values may be a one or two order delta of the previous value
- transformation of values (typically just to reduce their character length, since the format is plain-text)

Format Grammar

The version 1 (`v1`) format grammar is defined here. **Note: whitespace is not permitted in the format.**

`string`

- `version ':' all_values`
- `version ',' parameters ':' all_values`

`version`

- `'v' pos_integer`

`parameters`

- `parameter`
- `parameter ',' parameters`

`parameter`

- `delta`
- `blank`
- `scalar`
- `offset`

`delta`

- `'d0'`
- `'d1'`

- 'd2'

blank

- 'br'
- 'bn'
- 'b' number

scalar

- '*' number
- '/' number

offset

- '+' number
- '-' number

all_values

- ""
- nonblank_value
- nonblank_value ',' values

values

- value
- value ',' values

value

- blank_value
- nonblank_value

blank_value

- ""
- 'n' pos_integer

nonblank_value

- 'n'
- number
- number 'n' pos_integer
- 'nn' pos_integer

number

- integer fraction exponent

integer

- pos_integer

- neg_integer

pos_integer

- digit
- onenine digits

neg_integer

- '-' digit
- '-' onenine digits

digits

- digit
- digit digits

digit

- '0'
- onenine

onenine

- '1' . '9'

fraction

- ""
- '.' digits

exponent

- ""
- 'E' sign digits
- 'e' sign digits

sign

- ""
- '+'
- '-'

Version

The version denotes the encoded format for backwards compatibility, in case of future updates or changes. If the string does not include the version, it is assumed to be version 0, which is detailed at the end of this document.

Blanks

The `blank` parameter indicates how blank (empty) values should be interpretted. `'br'` (the default) repeats the previous encoded literal value. `'b' number` assigns a static numeric value. `'bn'` assigns the `null` value. `'br'` will be the most efficient in most cases, unless the majority of a data set is a single value. The format is considered invalid if more than one blank parameter is specified.

Given the data set:

- `0, 0, 1, 1, 1, 2, 2`

A `br` encoding would be:

- `v1,br,d0:0,,1,,,2,`

A `b0` encoding would be:

- `v1,b0,d0:0,,1,1,1,2,2`

Which can be further reduced to:

- `v1,b0,d0:0,,1n3,2,2`

Note, in this example, the first value must still be explicitly specified, per the first value non-blank rule.

Delta

The `delta` parameter defines the behavior for storing values as changing relative to the previous value. This can either be `'d0'` (disabling the feature), `'d1'` (the default, change from previous value), or `'d2'` (change of change from previous value). The format is considered invalid if more than one delta parameter is specified.

Given the simple data set:

- `0, 1, 2, 3, 4, 5, 6, 7, 8, 9`

The `d1` encoding would be:

- `v1,br,d1:0,1,1,1,1,1,1,1,1,1`

Which, with `br`, can be reduced to:

- `v1,br,d1:0,1,,,,,,,,`

Which can be further reduced to

- `v1,br,d1:0,1n9`

The `d2` encoding would be:

- `v1,br,d2:0,1,0,0,0,0,0,0,0,0`

Which can be further reduced to:

- `v1,br,d2:0,1,0n8`

Note that in `d1` encoding, the first value is not a delta but a static value. Likewise, in `d2`, the first value is static, and the second is the delta relative to the first.

When dealing with `null` values, the delta rules simply ignore any `null` values, carrying state from the previous non-`null` value. For example, given the data set:

- `null, null, null, 0, 1, 2, 3, 4, null, 5, 6, 7, 8, 9`

The `d1` encoding would be:

- `v1,br,d1:n,n,n,0,1,1,1,1,n,1,1,1,1,1`

Which can be further reduced to

- `nn3,0,1n4,n,1n5`

Delta Considerations for Fractional Values

When using the delta feature with fractional values, errors may be introduced into the data, depending on the encoding and decoding environments and implementation, particularly with regard with how floating point data is handled. Because each value is computed based on the previous on, the potential for error increases across the data set. For this reason, when working with fractional values it is strongly recommended to used a fixed precision context, by chosing an adequate static precision at encode time and embedding it with the scalar value, described below. Alternatively, the file can be encoded with the `d0` delta setting, disabling the feature. This can still provide significant compression benefit if the data set contains many repeated values.

Scalar / Offset

The `scalar` and `offset` parameters allow repetitive data to be further reduced. A single scalar value can be specified to either multiply (`'*'`) or divide (`'/'`) each value by, followed by a single positive (`'+'`) or negative (`'-'`) offset value. The scalar is always applied first, followed by the offset. The format is considered invalid if more than one scalar or more than one offset is present. These represent the operations to perform at decode time, so they are inversed during encoding. Note that a `/0` scalar is invalid.

For example, given the data set:

`50.0, 50.1, 50.3, 50.2, 49.9, 50.1`

We can use a /10 scalar and +50 offset to encode as follows:

`v1,d1,/10,+50:0,1,2,-1,-3,2`

Note that each `delta` operation is relative to the previous untransformed value.

Values

There are six possible values:

- `""` (empty, `blank`)
- `'n'` (`null` literal)
- `number`

- `'n' pos_integer` (`blank` repeated `integer` times)
- `number 'n' pos_integer` (number repeated `integer` times)
- `'nn' pos_integer` (`null` repeated `integer` times)

Note that the first value of the full value set must not be blank (or repeated blanks), to preserve the ability to define an empty data set. That is, the strings:

`"v1:"`, `"v1,b0:"`

Both denote a data set with no values.

Decoding Order of Operations

- Blank resolution
- Expansion (`n` repeated static values)
- Delta operations
- Scalar transform
- Offset transform

Implementation

We provide a reference encode / decode implementation in Java, TypeScript, and Python. Because the encoding process is very fast, multiple formats can usually be tested to find a reasonably optimal solution automatically.

General Recommendations

- Print values as minimally as possible (ie, `1e3` instead of `1000` or `1e+3`, `0` instead of `0.00`)
- When encoding fractional values with `delta` enabled, require an explicit scale input setting to inform the `scalar` parameter and prevent accumulated errors
- Avoid use of regular expressions in decoders, they often add considerable overhead and are overkill for the simplicity of the format

Changelog

Version 1

- **Breaking change**: In v0, `'n' integer` repeats previous value `integer` times. In v1, `'n' integer` repeats `blank` value `integer` times.
- **Breaking change**: Added support for `null` values
- **Breaking change**: Added support for `d0` (disabled delta mode), changed names from `d` to `d1`, `dd` to `d2`
- Added version specification to header
- Clarified parameter definition and limitations

Version 0

- Initial release

For reference the v0 grammar (note, number, scalar, and offset formats remain unchanged from v1 grammar):

`v0_string`

- `v0_parameters ':' v0_all_values`

`v0_parameters`

- `v0_parameter`
- `v0_parameter ',' v0_parameters`

`v0_parameter`

- `v0_delta`
- `v0_blank`
- `scalar`
- `offset`

`v0_delta`

- `'d'`
- `'dd'`

`v0_blank`

- `'br'`
- `'b' number`

`v0_all_values`

- `""`
- `number`
- `number ',' v0_values`

`v0_values`

- `v0_value`
- `v0_value ',' v0_values`

`v0_value`

- `""`
- `number`
- `v0_repeater`

`v0_repeater`

- `'n' pos_integer`

Revision #17
Created 11 February 2026 20:09:19 by Nick Dobson
Updated 1 July 2026 03:16:40 by Nick Dobson