Designing Message Styling

XEP-0393 and Mellium

Sam Whited

Formerly XSF Editor / Council

JID: sam@samwhited.com

2021–03–26

XEP-0286: Mobile Considerations on LTE Networks

XEP-0364: Current Off-the-Record Messaging Usage

XEP-0366: Entity Versioning

XEP-0367: Message Attaching

XEP-0375: XMPP Compliance Suites 2016

XEP-0377: Spam Reporting

XEP-0378: OTR Discovery

XEP-0383: Burner JIDs

XEP-0387: XMPP Compliance Suites 2018

XEP-0389: Extensible In-Band Registration

XEP-0393: Message Styling

XEP-0438: Best practices for password hashing and storage

XEP-0451: Stanza Multiplexing

XEP-0286: Mobile Considerations on LTE Networks

XEP-0364: Current Off-the-Record Messaging Usage

XEP-0366: Entity Versioning

XEP-0367: Message Attaching

XEP-0375: XMPP Compliance Suites 2016

XEP-0377: Spam Reporting

XEP-0378: OTR Discovery

XEP-0383: Burner JIDs

XEP-0387: XMPP Compliance Suites 2018

XEP-0389: Extensible In-Band Registration

XEP-0393: Message Styling

XEP-0438: Best practices for password hashing and storage

XEP-0451: Stanza Multiplexing

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

XEP-0393: Message Styling defines a formatted text syntax for use in instant messages with simple text styling.

Inline Styles

*emphasis*
strong emphasis
~~\~strike through\~~
`` `preformatted` ``

Preformatted Text

```optional-tag
                    __                                  _   _                 _
 _ __   _ __ ___   / _| ___  _ __ _ __ ___   __ _| |_| |_ ___  __| |
| '_ \| '__/ _ \ |_ / _ \| '__| '_ ` _ \ / _` | __| __/ _ \ / _` |
| |_) | | |  __/  _| (_) | |  | | | | | | (_| | |_| ||  __/ (_| |
| .__/|_|  \___|_|  \___/|_|  |_| |_| |_|\__,_|\__|\__\___|\__,_|
|_|

```

Block Quote

```
> This is a quote.
> It can have many lines.
>
>> It can even be nested!
```

Block Quote

> “This is a quote. It can have many lines.”
>
> > “It can even be nested!”

Block Quote

> This is a quote. It can have many lines.
>
> > It can even be nested!

Design Considerations

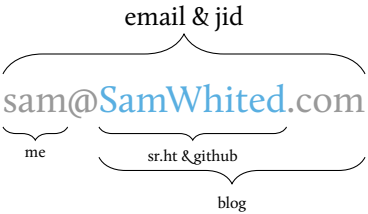
WhatsApp[1]

https://faq.whatsapp.com/general/chats/how-to-format-your-messages/

WhatsApp[1]

https://faq.whatsapp.com/general/chats/how-to-format-your-messages/

[1] Not Markdown.


email & jid
sam@SamWhited.com
me
sr.ht & github
blog

https://blog.samwhited.com/2020/11/message-styling/

https://mellium.im

`mellium.im/xmpp/styling`

Requirements

- Not tied to any specific markup
- No pathological edge cases
- Does not require parsing the entire document
- Must be able to distinguish directives from text

Requirements

- Not tied to any specific markup
- No pathological edge cases
- Does not require parsing the entire document
- Must be able to distinguish directives from text

Requirements

- Not tied to any specific markup
- No pathological edge cases
- Does not require parsing the entire document
- Must be able to distinguish directives from text

Requirements

- Not tied to any specific markup
- No pathological edge cases
- Does not require parsing the entire document
- Must be able to distinguish directives from text

Requirements

- Not tied to any specific markup
- No pathological edge cases
- Does not require parsing the entire document
- Must be able to distinguish directives from text

```
type Decoder
        func NewDecoder(r io.Reader) *Decoder
        func (*Decoder) Quote() uint
        func (*Decoder) SkipBlock() error
        func (*Decoder) SkipSpan() error
        func (*Decoder) Style() Style
        func (*Decoder) Token() (Token, error)
type Style
type Token
        func (Token) Copy() Token
```

```
type Decoder
    func NewDecoder(r io.Reader) *Decoder
    func (*Decoder) Quote() uint
    func (*Decoder) SkipBlock() error
    func (*Decoder) SkipSpan() error
    func (*Decoder) Style() Style
    func (*Decoder) Token() (Token, error)
type Style
type Token
    func (Token) Copy() Token
```

```
type Decoder
        func NewDecoder(r io.Reader) *Decoder
        func (*Decoder) Quote() uint
        func (*Decoder) SkipBlock() error
        func (*Decoder) SkipSpan() error
        func (*Decoder) Style() Style
        func (*Decoder) Token() (Token, error)
type Style
type Token
        func (Token) Copy() Token
```

```
type Decoder
        func NewDecoder(r io.Reader) *Decoder
        func (*Decoder) Quote() uint
        func (*Decoder) SkipBlock() error
        func (*Decoder) SkipSpan() error
        func (*Decoder) Style() Style
        func (*Decoder) Token() (Token, error)
type Style
type Token
        func (Token) Copy() Token
```

```
type Decoder
        func NewDecoder(r io.Reader) *Decoder
        func (*Decoder) Quote() uint
        func (*Decoder) SkipBlock() error
        func (*Decoder) SkipSpan() error
        func (*Decoder) Style() Style
        func (*Decoder) Token() (Token, error)
type Style
type Token
        func (Token) Copy() Token
```

```
type Decoder
        func NewDecoder(r io.Reader) *Decoder
        func (*Decoder) Quote() uint
        func (*Decoder) SkipBlock() error
        func (*Decoder) SkipSpan() error
        func (*Decoder) Style() Style
        func (*Decoder) Token() (Token, error)
type Style
type Token
        func (Token) Copy() Token
```

```
type Decoder
        func NewDecoder(r io.Reader) *Decoder
        func (*Decoder) Quote() uint
        func (*Decoder) SkipBlock() error
        func (*Decoder) SkipSpan() error
        func (*Decoder) Style() Style
        func (*Decoder) Token() (Token, error)
type Style
type Token
        func (Token) Copy() Token
```

```
type Decoder
        func NewDecoder(r io.Reader) *Decoder
        func (*Decoder) Quote() uint
        func (*Decoder) SkipBlock() error
        func (*Decoder) SkipSpan() error
        func (*Decoder) Style() Style
        func (*Decoder) Token() (Token, error)
type Style
type Token
        func (Token) Copy() Token
```

Iterating

```
r := strings.NewReader("It's called _*Twelfth Night*_.")
d := styling.NewDecoder(r)
for {
  tok, err := d.Token()
  if errors.Is(err, io.EOF) {
    break
  }
  if err != nil {
    return err
  }
  // …
}
```

```
type Decoder
        func NewDecoder(r io.Reader) *Decoder
        func (*Decoder) Quote() uint
        func (*Decoder) SkipBlock() error
        func (*Decoder) SkipSpan() error
        func (*Decoder) Style() Style
        func (*Decoder) Token() (Token, error)
type Style
type Token
        func (Token) Copy() Token
```

```
type Decoder
        func NewDecoder(r io.Reader) *Decoder
        func (*Decoder) Quote() uint
        func (*Decoder) SkipBlock() error
        func (*Decoder) SkipSpan() error
        func (*Decoder) Style() Style
        func (*Decoder) Token() (Token, error)
type Style
type Token
        func (Token) Copy() Token
```

```
type Token
        func (Token) Copy() Token
```

```go
// A Token represents a styling directive or unstyled span of text.
// If the token is a preformatted text block start token, Info is a
// slice of the bytes between the code fence (```) and the newline.
type Token struct {
  Mask Style
  Data []byte
  Info []byte
}

// Copy creates a new copy of the token.
func (t Token) Copy() Token
```

```
// Style is a bitmask that represents a set of styles that can be
// applied to text.
type Style uint32
```

```
// Style is a bitmask that represents a set of styles that can be
// applied to text.
type Style uint32

const (
  BlockPre Style = 1 << iota
  BlockQuote
  SpanEmph
  SpanStrong
  SpanStrike
  SpanPre

  // …
)
```

```
// Style is a bitmask that represents a set of styles that can be
// applied to text.
type Style uint32

const (
  // …

  BlockPreStart
  BlockPreEnd
  BlockQuoteStart
  BlockQuoteEnd

  // …
)
```

```
// Style is a bitmask that represents a set of styles that can be
// applied to text.
type Style uint32

const (
  // …

  SpanEmphStart
  SpanEmphEnd
  SpanStrongStart
  SpanStrongEnd

  // …
)
```

```
// Style is a bitmask that represents a set of styles that can be
// applied to text.
type Style uint32

const (
  // …

  SpanStrikeStart
  SpanStrikeEnd
  SpanPreStart
  SpanPreEnd

  // …
)
```

```
// Style is a bitmask that represents a set of styles that can be
// applied to text.
type Style uint32

const (
  // …

  Block                = BlockPre | BlockQuote
  Span                 = SpanEmph | SpanStrong | SpanStrike | SpanPre
  SpanEmphDirective    = SpanEmphStart | SpanEmphEnd
  SpanStrongDirective = SpanStrongStart | SpanStrongEnd

  // …
)
```

Using Bitwise Operations on Styles

```
if tok.Mask&styling.Directive == styling.Directive {
  /* lower contrast */
}
```

Using Bitwise Operations on Styles

```
switch {
case tok.Mask&styling.SpanStrongStart == styling.SpanStrongStart:
  out.WriteString("<strong><code>")
  out.Write(tok.Data)
  out.WriteString("</code>")
case tok.Mask&styling.SpanStrongEnd == styling.SpanStrongEnd:
  out.WriteString("<code>")
  out.Write(tok.Data)
  out.WriteString("</code></strong>")
// …
}
```

Possible Future QoL Enhancement:

```
// Directive checks if the style is any styling directive.
func (s Style) Directive() bool {
  return s&Directive == Directive
}

// Emph checks if the style includes emphasis.
func (s Style) Emph() bool {
  return s&SpanEmph == SpanEmph
}
```

```
type Decoder
        func NewDecoder(r io.Reader) *Decoder
        func (*Decoder) Quote() uint
        func (*Decoder) SkipBlock() error
        func (*Decoder) SkipSpan() error
        func (*Decoder) Style() Style
        func (*Decoder) Token() (Token, error)
type Style
```

```
type Decoder
        func NewDecoder(r io.Reader) *Decoder
        func (*Decoder) Quote() uint
        func (*Decoder) SkipBlock() error
        func (*Decoder) SkipSpan() error
        func (*Decoder) Style() Style
        func (*Decoder) Token() (Token, error)
type Style
```

```
type Decoder
        func NewDecoder(r io.Reader) *Decoder
        func (*Decoder) Quote() uint
        func (*Decoder) SkipBlock() error
        func (*Decoder) SkipSpan() error
        func (*Decoder) Style() Style
        func (*Decoder) Token() (Token, error)
type Style
```

type Decoder
    func NewDecoder(r io.Reader)
*Decoder
    func (*Decoder) Quote() uint
    func (*Decoder) SkipBlock() error
    func (*Decoder) SkipSpan() error
    func (*Decoder) Style() Style
    func (*Decoder) Token() (Token,
error)
type Style

```
type Token struct {
  Mask Style
  Data []byte
  Info []byte
}
```

Iterating

```
r := strings.NewReader("It's called _*Twelfth Night*_.")
d := styling.NewDecoder(r)


for {
  tok, err := d.Token()
  if errors.Is(err, io.EOF) {
    break
  }
  if err != nil {
    return err
  }
  // …
}
```

Iterating

```
r := strings.NewReader("It's called _*Twelfth Night*_.")
d := styling.NewDecoder(r)
```

```
for {
  tok, err := d.Token()
  if errors.Is(err, io.EOF) {
    break
  }
  if err != nil {
    return err
  }
  // …
}
```

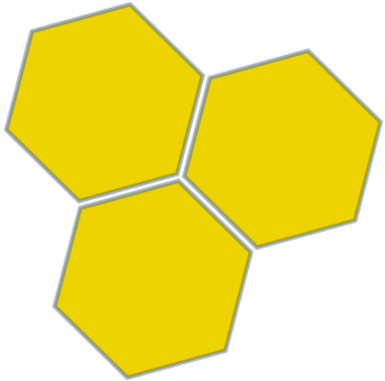
```
for d.Next() {
  tok := d.Token()
  // …
}
if d.Error() != nil {
  return err
}
```

Chat Rooms

`go@gopher.chat`

Chat Rooms

go@gopher.chat



users@mellium.chat

email & jid

sam@SamWhited.com

me

sr.ht & github

blog

go@gopher.chat

users@mellium.chat