Improving event structure for human representation and handling unrecognised event types.

This is now obsoleted by https://docs.google.com/document/d/1FUVFzTOF4a6XBKVTk55bVRIk4N9u5uZkHS4Rjr_SGxo

The problem:

1. we don't have a way of expressing arbitrary event types in an optional human representation for display in a timeline (eg a chat client, debugging logs, or other visualisation of events to be consumed by humans)
2. we don't have a way of handling type hierarchies of **unknown** event types - eg knowing that a "video" should be treated like a "file" if our client cannot handle video specifically.
3. we don't have a way of annotating human representations to show which underlying data fields they relate to. **<- the options below do not try to solve this issue**

(this is an attempt at a simpler re-statement of the problem & current solutions on the table from the more sprawling and disorganised https://docs.google.com/document/d/1l08DL_F_CHo1pIORXzcqWLIZQpztogMFUTYmxd4Gr5s/edit)

I like Option G atm (Feb 2018) -- matthew

Option A: Any event can have body/_display keys for human representation.

Pros: Direct backwards compat. Simple. No server changes.
Cons: Doesn't address event specialisation (e.g. "this event describes a movie, which is a type of file, which is a type of room message") or collation ("this event describes both temperature and humidity data types")

example 1:

```
{
    type: "net.arasphere.temperature", // the matrix event.type
    content: {
        body: "The temperature is 37C",
        _display: [
            {
                mimetype: "text/html",
                body: "The temperature is <font color='#f00'>37C</font>"
            },
            {
                mimetype: "text/markdown",
                body: "The temperature is **37C**"
```

```
            },
        ],
        temperature: 37.0,
    }
}
```

example 2:

```
{
    type: "m.room.video",
    content: {
        body: "filename.mov",
        _display: [
            {
                mimetype: "text/html",
                body: "Video of mxc://matrix.org/378ye12§wdqefdv",
            }
        ],
        url: "mxc://matrix.org/378ye12§wdqefdv",
        size: 6413423,
        width: 640,
        height: 480,
        duration: 350,
    }
}
```

Option B: We collate together events which describe the equivalent thing

Pros: supports groupings of arbitrary events. no risk of clashes between fields of events.
Cons: Server & Client changes; not backwards compatible; doesn't easily allow metadata from one event to be referenced in another. Gets confused with other 'relates to' mechanisms to associate events with one another.

<Rejected as disadvantages too significant>

```
{
    type: "net.arasphere.ambient",
    content: {
        subevents: [
            {
                type: "net.arasphere.temperature",
                temperature: 37.0,
                body: "Temperature is 37C",
                _display: ...,
            },
            {
                type: "net.arasphere.humidity",
                humidity: 68,
                body: "Humidity is 68%",
```

```
                _display: ...,
            }
        ]
    }
}

(human representation is "Temperature is 37C, Humidity is 68%")
```

Option C: We define a type hierarchy on each event

(as proposed by Aviral). Client tries to present UI for the most specific type it can. (Mixins is a subset of this)

Pros: Can be backwards compatible. supports specialisation & collations of arbitrary events.
Cons: Minor Server & Client changes. risk of clashes between field names. confusing behaviour if parent types get missed out, or if you specify 2 parent types which have the same key name (multiple inheritance!!).

```
{
    type: "net.arasphere.temperature", // the matrix event.type
    extends: [ "m.room.message" ],
    content: {
        body: "The temperature is 37C", // from m.room.message
        _display: [                     // from m.room.message
            {
                mimetype: "text/html",
                body: "The temperature is <font color='#f00'>37C</font>"
            },
            {
                mimetype: "text/markdown",
                body: "The temperature is **37C**"
            },
        ],
        temperature: 37.0, // from net.arasphere.temperature
    }
}
```

or:

```
{
    type: "m.room.video",
    extends: [ "m.room.file", "m.room.message" ],
    content: {
        url: "mxc://matrix.org/378ye12§wdqefdv", // taken from m.room.file
        size: 6413423,  // taken from m.room.file
        width: 640,     // taken from m.room.video
```

```
        height: 480,    // taken from m.room.video
        duration: 350,  // taken from m.room.video
    }
}
```

Option D: Duck Typing

Pros: solves event combination. no risk of clashes between field names. vaguely backwards compatible (as type is effectively ignored).
Cons: longer field names; more bandwidth. implementation-wise it'd be nice to have a meaningful 'type' field rather than having to guess from the ungodly mix of fields on the event - and this doesn't even work on E2E, so we'd have to have the 'extends' thing anyway.

```
{
    type: "m.room.message",
    content: {
        net.arasphere.temperature.temperature: 37.0,
        body: "The temperature is 37C",
        body.text/html: "The temperature is <font color='#f00'>37C</font>",
        body.text/markdown: "The temperature is **37C**"
    }
}
```

or:

```
{
    type: "m.room.message",
    content: {
        m.room.file.url: "mxc://matrix.org/378ye12§wdqefdv", // taken from m.room.file
        m.room.file.size: 6413423,  // taken from m.room.file
        m.room.video.width: 640,    // taken from m.room.video
        m.room.video.height: 480,   // taken from m.room.video
        m.room.video.duration: 350, // taken from m.room.video
    }
}
```

Option E: Put event hierarchy in the event type.

Pros: solves event specialisation (but not combination). vaguely backwards compatible.
Cons: doesn't provide arbitrary mixins or event collation.

```
{
    // we keep lopping off the final .$term sequence of the type
    // until we find a type we know how to display
    type: "m.room.message.file.video",
    content: {
        body: "filename.mov",
        _display: [
            {
                mimetype: "text/html",
                body: "Video of mxc://matrix.org/378ye12$wdqefdv",
            }
        ],
        url: "mxc://matrix.org/378ye12$wdqefdv",
        size: 6413423,
        width: 640,
        height: 480,
        duration: 350,
    }
}
```

Option F: Mixins but with explicit types and grouping

Pros: solves event specialisation, combination, arbitrary mixins
Cons: not backwards compatible, although one can keep the old keys in there.
We deliberately don't allow for multiple 'attachments' for a given message, and instead assume that relates_to would let you associate such attachments with a parent message. Likewise, no nesting. This is just a way of mixing in different types to have a richer *single* event.

```
{
    type: "m.room.image",
    also: ["m.room.location", "m.room.message", "m.room.file"],
    content: {
        m.body: {
            text/plain: "A beautiful image",
            text/html: "A <i>beautiful</i> image",
        },
        m.thumbnail: {
            w: 160,
            h: 120,
            mimetype: 'image/jpg',
            url: 'mxc://matrix.org/21fhuiwvuihs'
        },
        m.location: "geo:37.786971,-122.399677;u=35",
        m.image: {
            w: 640,
            h: 480,
        },
        m.file: {
            url: 'mxc://matrix.org/28zouhcxiuhe'
            mimetype: 'image/jpg',
            size: 542321,
        }
    }
}
```

Option G: Like option F, except that the fields are under the type.

(by Erik)

```
{
        "type":  "m.room.message",
        "content": {
                "m.room.location": { "uri": "geo:37.786971,-122.399677;u=35"},
                "m.room.message": { … },
                "m.room.file": {
                        "url: "mxc://matrix.org/28zouhcxiuhe",
                        "mimetype": "image/jpg",
                       "size": 542321,
                },
                "m.room.image": {
                        "w": 640,
                        "h": 480,
                },
                "m.room.thumbnail": {
                        "w”: 160,
                        "h": 120,
                        “size”: 1234,
                        "mimetype": "image/jpg",
                        "url": "mxc://matrix.org/21fhuiwvuihs"
                 }
        }
}
```

This is mainly an example of a way of doing something that is easier for non-dynamic language clients to consume. For example, this event could easily be serialized/deserialized as something like:

{content: Map<String, MessageType>}

Where MessageType is an algabraic enum like:

```
{
        Location{ uri: String },
        Message { body: String, ….}
        …
}
```