

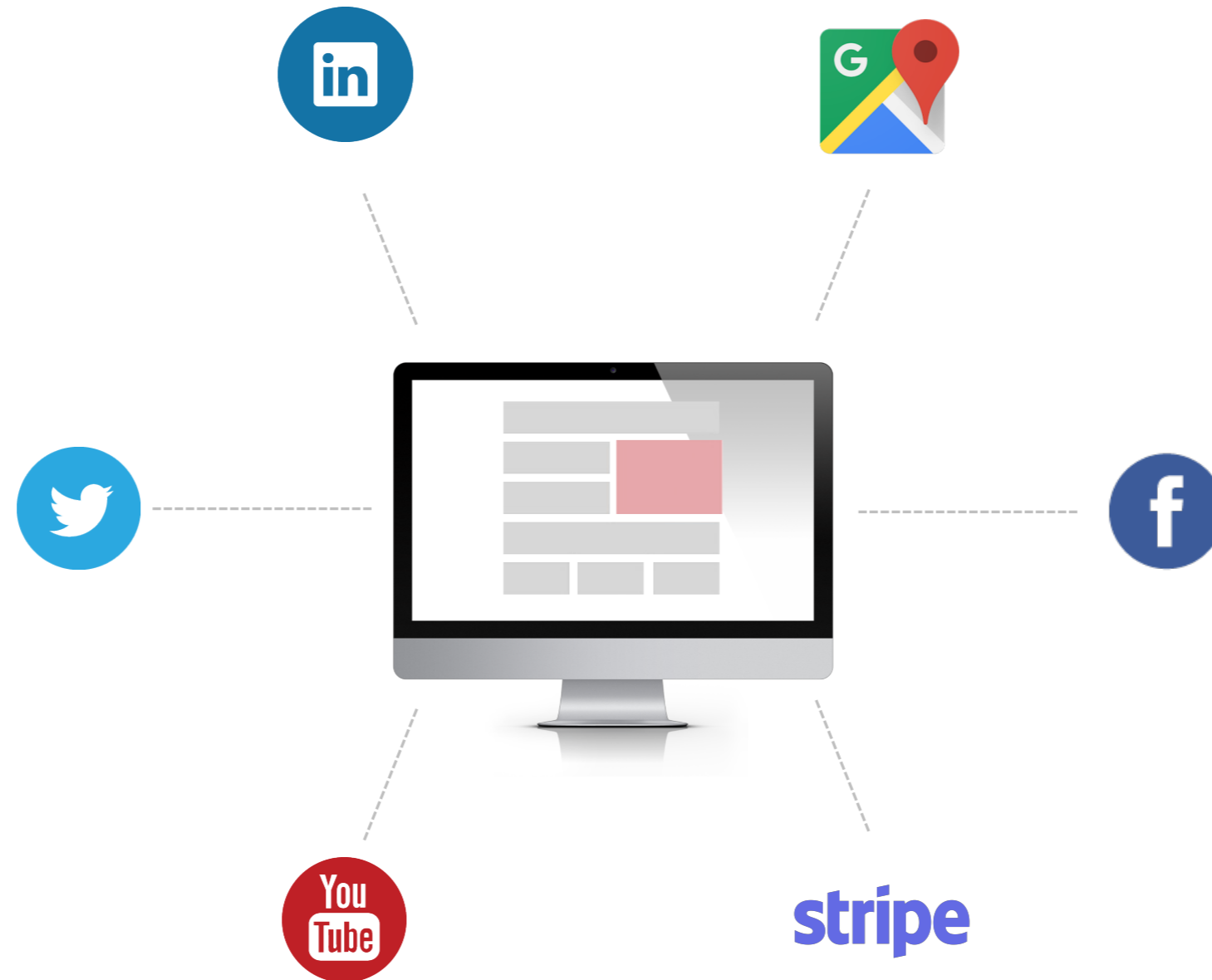


Complex Presence Constraints in Web Applications

Deliverable 4.1.1

Nathalie Oostvogels

Third party web services



Web API specifications



Parameters

One of `user_id` or `screen_name` are required.

`user_id`
optional

optional

number

Example Values: 12345

`screen_name`
optional

optional

string

Example Values: noradio

`text`
required

required

string

Example Values:

Meet me behind the cafeteria after school

Verifying the constraints

```
export module Twitter{  
  export interface PrivateMessage {  
    user_id?: number;  
    screen_name?: string;  
    text: string;  
  }  
  
  export function sendPrivateMessage(body: PrivateMessage){  
    // ...  
  }  
}
```

Web API specifications



Parameters

One of `user_id` or `screen_name` are required.

`user_id`
optional

optional

number

Example Values: 12345

`screen_name`
optional

optional

string

Example Values: noradio

`text`
required

required

string

Example Values:

Meet me behind the cafeteria after school

Web API specifications

✘ “Incompatible parameters specified in the request”

✘ “Some co-ordinate parameters were blank”

✘ “You must specify either a list ID or a slug and owner”

Web API specifications



Parameters

One of `user_id` or `screen_name` are required.

`user_id`
optional

optional

number

Example Values: 12345

`screen_name`
optional

optional

string

Example Values: noradio

`text`
required

required

string

count limit (available in the [help/configuration](#) endpoint)

Example Values:

Meet me behind the cafeteria after school

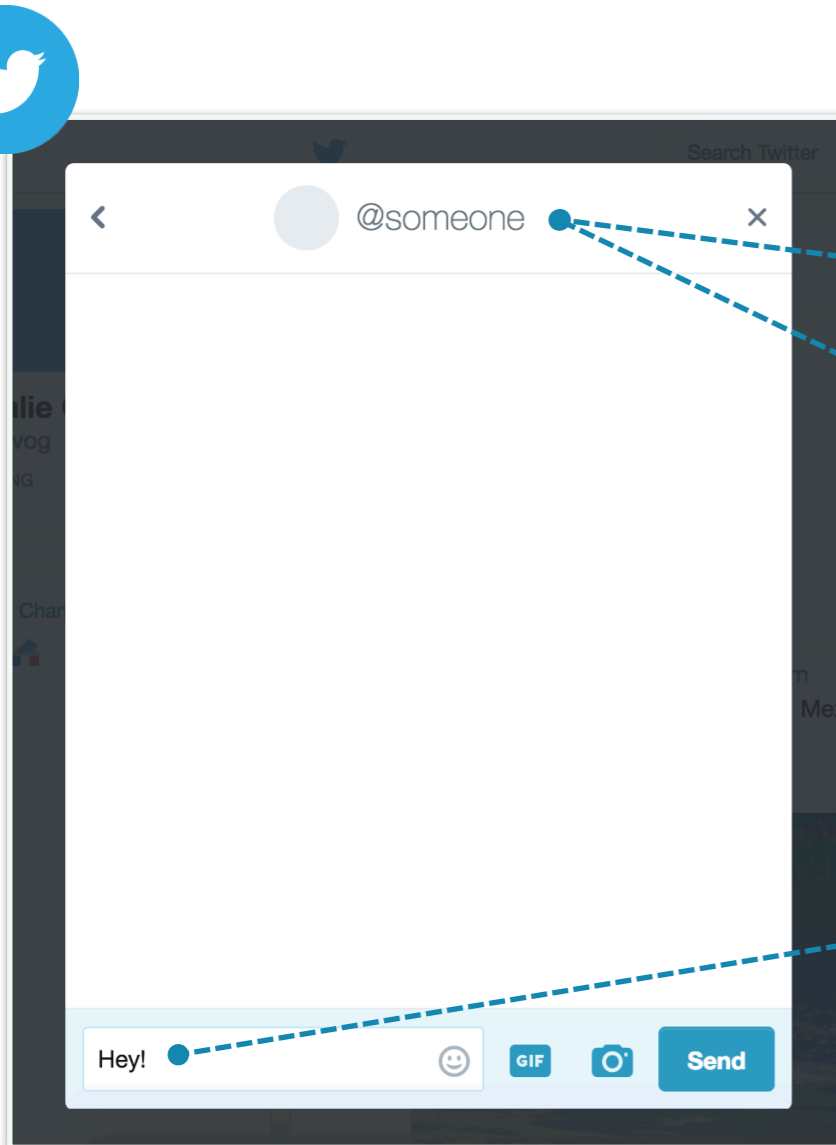
Either `user_id`
or
`screen_name`



Inter-parameter constraints

Exclusive constraints

➔ exactly one of a set of parameters is required



Parameters

One of `user_id` or `screen_name` are required.

`user_id`
optional

The ID of the user who should receive the direct message. Helpful for disambiguating when a valid user ID is also a valid screen name.

Example Values: 12345

`screen_name`
optional

The screen name of the user who should receive the direct message. Helpful for disambiguating when a valid screen name is also a user ID.

Example Values: noradio

`text`
required

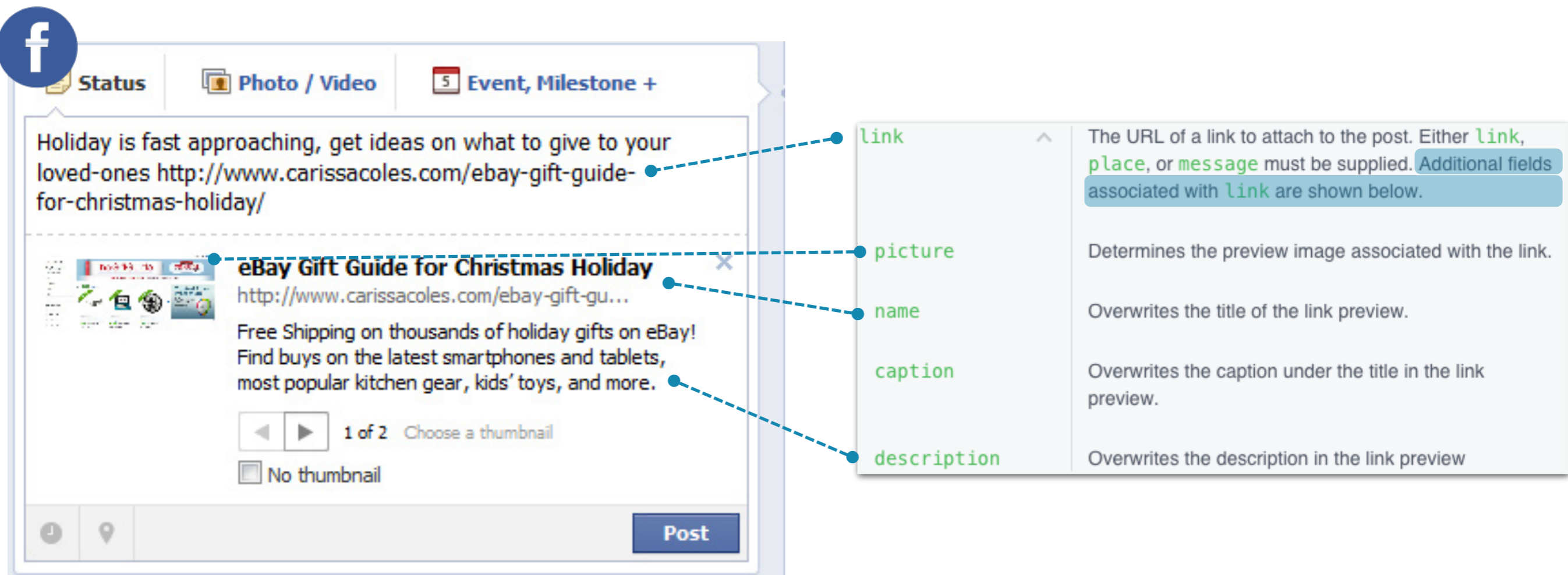
The text of your direct message. Be sure to URL encode as necessary, and keep the message within the character count limit (available in the [help/configuration](#) endpoint)

Example Values:

Meet me behind the cafeteria after school

Dependent constraints

➔ constraints on a parameter **depend** on a property of another parameter



The image shows a Facebook post creation interface. The post text is "Holiday is fast approaching, get ideas on what to give to your loved-ones <http://www.carissacoles.com/ebay-gift-guide-for-christmas-holiday/>". Below the text is a link preview for "eBay Gift Guide for Christmas Holiday" with a thumbnail image and a description: "Free Shipping on thousands of holiday gifts on eBay! Find buys on the latest smartphones and tablets, most popular kitchen gear, kids' toys, and more." The interface includes a "Post" button and a "Choose a thumbnail" section with "1 of 2" thumbnails and a "No thumbnail" checkbox.

The legend on the right lists the following parameters and their descriptions:

- link**: The URL of a link to attach to the post. Either **link**, **place**, or **message** must be supplied. Additional fields associated with **link** are shown below.
- picture**: Determines the preview image associated with the link.
- name**: Overwrites the title of the link preview.
- caption**: Overwrites the caption under the title in the link preview.
- description**: Overwrites the description in the link preview.

Dependent constraints

➔ constraints on a parameter **depend** on a property of another parameter



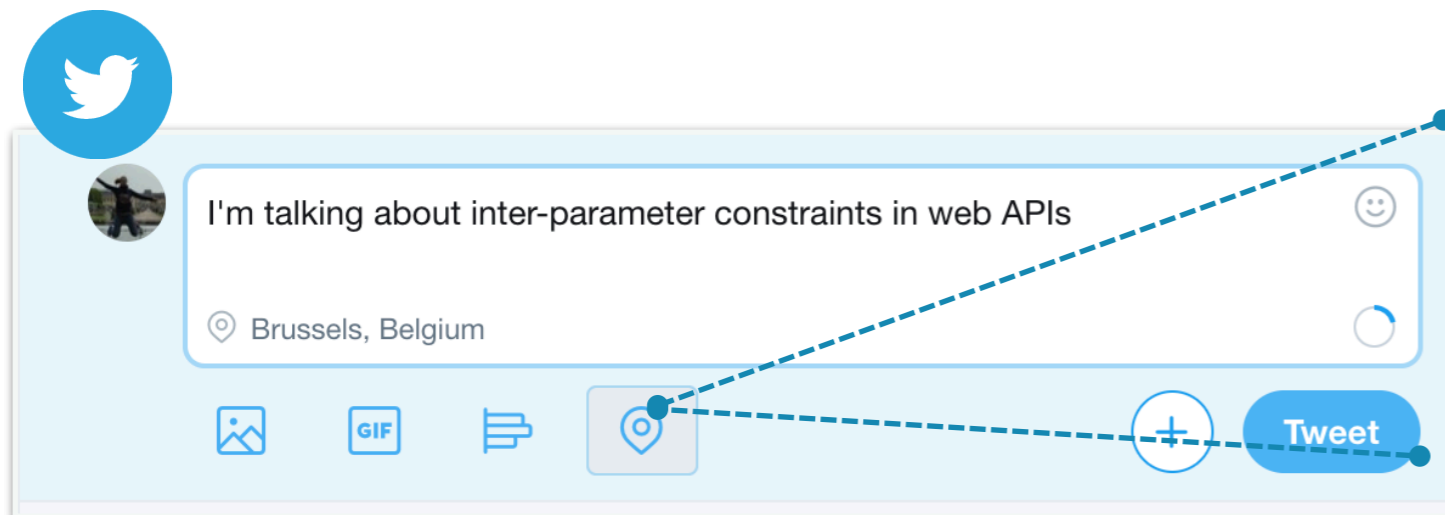
the property **infoWindow** is ignored, when **suppressInfoWindows** is “true”



when searching for an item, **condition** cannot be set to “new” when the **availability** parameter is set to “available”.

Group constraints

- ➔ a set of parameters should either be all excluded from a request or all included



lat
optional

The latitude of the location this tweet refers to. This parameter will be ignored unless it is inside the range -90.0 to +90.0 (North is positive) inclusive. It will also be ignored if there isn't a corresponding `long` parameter.

Example Values: 37.7821120598956

long
optional

The longitude of the location this tweet refers to. The valid ranges for longitude is -180.0 to +180.0 (East is positive) inclusive. This parameter will be ignored if outside that range, if it is not a number, if `geo_enabled` is disabled, or if there not a corresponding `lat` parameter.

Example Values: -122.400612831116

Inter-parameter constraints are everywhere



exactly one of `user_id` and `screen_name` is required



the property `infoWindow` is ignored,
when `suppressInfoWindows` is “true”



`latitude` will be ignored when there is no
corresponding `longitude` parameter



it is an error to specify values for both `times` and `ns`



the `InterfaceId` or `InterfaceIndex` [...] must be set
to the interface for which to retrieve information

inter-parameter constraints

in web applications

Verifying the constraints

```
export module Twitter{  
  export interface PrivateMessage {  
    user_id?: number;  
    screen_name?: string;  
    text: string;  
  }  
  
  export function sendPrivateMessage(body: PrivateMessage){  
    // ...  
  }  
}
```

Complex inter-property constraints



“You can identify a list by its **slug** instead of its **list_id**. If you decide to do so, note that you will also have to specify the list owner using the **owner_id** or **owner_screen_name** parameters.”

XOR

XOR

IPCs in programming languages

exactly one of *user_id* and *screen_name* is required



user_id optional
<hr/>
screen_name optional
<hr/>
text required

```
interface PrivateMessage {  
    user_id:      number;  
    screen_name:  string;  
    text:         string;  
} constraining {  
    present(text);  
    present(user_id) XOR present(screen_name);  
}
```



Programming with inter-property constraints

Initialisation

➔ object must satisfy constraints

```
present(text)
present(user_id) XOR present(screen_name)
```

```
var msg1: PrivateMessage = {text: "Hello",
                             user_id: 42};
```



```
var msg2: PrivateMessage = {text: "Hello"};
```



```
var msg3: PrivateMessage = {text: "Hello",
                             user_id: 42,
                             screen_name: "Alice"}
```



Accessing properties

➔ only allowed when property is known to be present or absent

```
function getInfo(msg: PrivateMessage) {  
    msg.text;                //:: string  
  
    msg.user_id;              
  
    msg.screen_name;          
  
}
```

Accessing properties

➔ if-tests provide more information about presence of properties

```
function getInfo(msg: PrivateMessage) {  
    msg.text;                //:: string  
  
    if (msg.user_id !== undefined) {  
        msg.user_id;        //:: number  
    } else {  
        msg.user_id;        //:: undefined  
    }  
}
```

Accessing properties

➔ if-tests provide more information about presence of properties

```
function getInfo(msg: PrivateMessage) {  
  
    msg.text;                //:: string  
  
    if (msg.user_id !== undefined) {  
        msg.user_id;        //:: number  
        msg.screen_name;    //:: undefined  
    } else {  
        msg.user_id;        //:: undefined  
        msg.screen_name;    //:: string  
    }  
  
}
```

Updating properties

```
function setInfo(msg: PrivateMessage) {
```

```
    msg.text = "Hello again";
```



```
    msg.user_id = 43;
```



```
    if (msg.user_id !== undefined) {
```

```
        msg.user_id = 43;
```



```
        msg.screen_name = undefined;
```




```
    }
```

```
}
```

 TypeScript strict
null checking mode

Updating multiple properties


```
var msg: PrivateMessage = {text: "Hello", user_id: 42};  
  
if (msg.user_id !== undefined) {  
  msg.user_id = undefined;  
  msg.screen_name = "Alice";  
}
```



Updating multiple properties

```
var msg1: PrivateMessage = {text: "Hello", user_id: 42};
```

```
var msg2 = assign(msg1, {user_id: undefined,  
                        screen_name: "Alice"});
```



```
var msg3 = assign(msg1, {user_id: undefined});
```



 all related properties need to be updated

Updating multiple properties

➔ constraints form clusters

```
interface PrivateMessage' {  
  text: string;  
  r_user_id: number;  
  r_screen_name: string;  
  s_user_id: number;  
  s_screen_name: string;  
} constraining {  
  present(text);  
  present(r_user_id) XOR  
    present(r_screen_name);  
  present(s_user_id) XOR  
    present(s_screen_name);  
}
```

```
var msg: PrivateMessage' =  
  {text: "Hello",  
   r_user_id: 42, s_user_id: 43};  
  
var msg2 = assign(msg,  
  {r_user_id: 44,  
   r_screen_name: undefined});
```

Updating multiple properties

➔ constraints form clusters

```
interface PrivateMessage' {  
  text: string;  
  r_user_id: number;  
  r_screen_name: string;  
  s_user_id: number;  
  s_screen_name: string;  
} constraining {  
  present(text);  
  present(r_user_id) XOR  
    present(r_screen_name);  
  present(s_user_id) XOR  
    present(s_screen_name);  
}
```

```
var msg: PrivateMessage' =  
  {text: "Hello",  
   r_user_id: 42, s_user_id: 43};  
  
var msg2 = assign(msg,  
  {r_user_id: 44,  
   r_screen_name: undefined});
```

❗ assign is functional

Verifying inter-property constraints

with propositional logic

Object literals have to satisfy constraints

```
var msg: PrivateMessage = {text: "Hello", user_id: 42};
```

valuation **v**: assigns **truth values** to **proposition letters**



presence or absence



property names


❗ right-hand side needs to be an object literal

Property Access

$$\{ \text{interface constraints} \} \models \text{present}(\text{property})$$

Constraints dictate property presence


➔ verify presence/absence with logical entailment

```
function getInfo(msg: PrivateMessage) {  
    msg.text;  //:: string  
  
    msg.user_id;  
}
```

$$\left\{ \begin{array}{l} \text{present(text)} \\ \text{present(screen_name) XOR present(user_id)} \end{array} \right\} \models \text{present(text)}$$

Constraints dictate property presence


⇒ verify presence/absence with logical entailment

```
function getInfo(msg: PrivateMessage) {  
    msg.text; //:: string  
  
    msg.user_id;   
}
```

$\left\{ \begin{array}{l} \text{present(text)} \\ \text{present(screen_name) XOR present(user_id)} \end{array} \right\} \not\models \text{present(user_id)}$

Constraints dictate property presence

⇒ verify presence/absence with logical entailment

```
function getInfo(msg: PrivateMessage) {  
    msg.text; //:: string  
  
    msg.user_id;   
}
```

$\left\{ \begin{array}{l} \text{present}(\text{text}) \\ \text{present}(\text{screen_name}) \text{ XOR } \text{present}(\text{user_id}) \end{array} \right\} \not\models \text{present}(\text{user_id}) \wedge \neg \text{present}(\text{user_id})$

Constraints dictate property presence


⇒ verify presence/absence with logical entailment

```
function getInfo(msg: PrivateMessage) {  
    msg.text; //:: string  
  
    if (msg.user_id !== undefined) {  
        msg.user_id;  
    }  
}
```

$$\left\{ \begin{array}{l} \text{present(text)} \\ \text{present(screen_name) XOR present(user_id)} \\ \text{present(user_id)} \end{array} \right\} \models$$

Constraints dictate property presence


⇒ verify presence/absence with logical entailment

```
function getInfo(msg: PrivateMessage) {  
    msg.text; //:: string  
  
    if (msg.user_id !== undefined) {  
        msg.user_id;  //:: number  
    }  
}
```

$$\left\{ \begin{array}{l} \text{present(text)} \\ \text{present(screen_name) XOR present(user_id)} \\ \text{present(user_id)} \end{array} \right\} \models \text{present(user_id)}$$

Constraints dictate property presence

⇒ verify presence/absence with logical entailment

```
function getInfo(msg: PrivateMessage) {  
    msg.text; //:: string  
  
    if (msg.user_id !== undefined) {  
        msg.user_id; //:: number  
        msg.screen_name;  //:: undefined  
    }  
}
```

$$\left\{ \begin{array}{l} \text{present(text)} \\ \text{present(screen_name) XOR present(user_id)} \\ \text{present(user_id)} \end{array} \right\} \models \neg \text{present(screen_name)}$$

Interface Compatibility

`var target = source`

$$\left\{ \begin{array}{l} \text{source constraints} \\ + \text{ structural delta} \end{array} \right\} \models \left\{ \begin{array}{l} \text{target constraints} \\ + \text{ structural delta} \end{array} \right\}$$

Assignment

```
interface PrivateMessage {  
  text: string;  
  user_id: number;  
  screen_name: string;  
} constraining {  
  present(text);  
  present(user_id) XOR  
    present(screen_name);  
}
```



```
interface PrivateMessageId {  
  text: string;  
  user_id: number;  
} constraining {  
  present(text);  
  present(user_id);  
}
```



```
var msg: PrivateMessage = ...;  
var msgId: PrivateMessageId = ...;  
msgId = msg;  
msg = msgId;
```

Assignment

```
interface PrivateMessage {  
  text: string;  
  user_id: number;  
  screen_name: string;  
} constraining {  
  present(text);  
  present(user_id) XOR  
    present(screen_name);  
}
```



```
interface PrivateMessageId {  
  text: string;  
  user_id: number;  
} constraining {  
  present(text);  
  present(user_id);  
}
```

```
var source: PrivateMessageId = {text: "Hello", user_id: 42};  
var target: PrivateMessage = source;
```

$\left\{ \begin{array}{l} \text{present(text)} \\ \text{present(user_id)} \end{array} \right\} \neq \begin{array}{l} \text{present(text)} \\ \text{present(user_id) XOR} \\ \text{present(screen_name)} \end{array}$

source: PrivateMessageId

target: PrivateMessage

Assignment

```
interface PrivateMessage {  
  text: string;  
  user_id: number;  
  screen_name: string;  
} constraining {  
  present(text);  
  present(user_id) XOR  
    present(screen_name);  
}
```



```
interface PrivateMessageId {  
  text: string;  
  user_id: number;  
} constraining {  
  present(text);  
  present(user_id);  
}
```

```
var source: PrivateMessageId = {text: "Hello", user_id: 42};  
var target: PrivateMessage = source;
```

$$\left\{ \begin{array}{l} \text{present(text)} \\ \text{present(user_id)} \\ \neg \text{present(screen_name)} \end{array} \right\} \models \begin{array}{l} \text{present(text)} \\ \text{present(user_id) XOR} \\ \text{present(screen_name)} \end{array}$$

source: PrivateMessageId

target: PrivateMessage

Updating multiple properties

```
interface PrivateMessage' {  
  text: string;  
  r_user_id: number;  
  r_screen_name: string;  
  s_user_id: number;  
  s_screen_name: string;  
} constraining {  
  present(text);  
  present(r_user_id) XOR  
    present(r_screen_name);  
  present(s_user_id) XOR  
    present(s_screen_name);  
}
```

```
var msg: PrivateMessage' =  
  {text: "Hello",  
   r_user_id: 42, s_user_id: 43};  
  
var msg2 = assign(msg,  
  {r_user_id: 44,  
   r_screen_name: undefined});
```

Updating multiple properties

```
interface PrivateMessage' {  
  text: string;  
  r_user_id: number;  
  r_screen_name: string;  
  s_user_id: number;  
  s_screen_name: string;  
} constraining {  
  present(text);  
  present(r_user_id) XOR  
    present(r_screen_name);  
  present(s_user_id) XOR  
    present(s_screen_name);  
}
```

```
var msg: PrivateMessage' =  
  {text: "Hello",  
   r_user_id: 42, s_user_id: 43};  
  
var msg2 = assign(msg,  
  {r_user_id: 44,  
   r_screen_name: undefined});
```

valuation



Future Work

➔ value-dependent constraints



the property `infoWindow` is ignored,
when `suppressInfoWindows` is *true*



if the `steppedLine` value is set to anything other than *false*,
`lineTension` will be ignored

Implementation

```
1 interface PrivateMessage {  
2   · text?: string;  
3   · userid?: number;  
4   · screenname?: string;  
5 } constrains {  
6   · present(text);  
7   · xor(present(userid), present(screenname));  
8 }  
9
```

```
10 • let msg: PrivateMessage = {text: "Hello", userid: 42, screenname: "Alice"};
```

Type '{ text: string; userid: number; screenname: string; }' is not assignable to type 'PrivateMessage'.

Predicate xor(present(userid),present(screenname)) was not satisfied in type { text: string; userid: number; screenname: string; }



<https://github.com/noostvog/TypeScriptIPC>

Revisiting the mock-up server

```
export module Twitter{
  export interface PrivateMessage {
    text: string;
    user_id: number;
    screen_name: string;
  }
  constraining {
    present(text);
    present(user_id) XOR present(screen_name);
  }

  export function sendPrivateMessage(body: PrivateMessage){
    // ...
  }
}
```

interfaces with inter-property constraints

propositional logic to define and verify constraints

Spotter guide to inter-property constraints:

Exclusive constraint:
either, exactly, subsumed or one of

Dependent constraint:
additional or providing

Group constraint:
corresponding or providing