```
class CPoint {
    x: number;
    y: number;
    constructor(x: number,  y: number) {
        this.x = x;
        this.y = y;
    }
}

getX(new CPoint(0, 0));  // Ok, fields match

getX({ x: 0, y: 0, color: "red" });  // Extra fields Ok

getX({ x: 0 });  // Error: supplied parameter does not match
```

See section 3.8 for more information about type comparisons.

1.5 Contextual Typing

Ordinarily, TypeScript type inference proceeds "bottom-up": from the leaves of an expression tree to its root. In the following example, TypeScript infers 'number' as the return type of the function 'mul' by flowing type information bottom up in the return expression.

```
function mul(a: number, b: number) {
    return a * b;
}
```

For variables and parameters without a type annotation or a default value, TypeScript infers type 'any', ensuring that compilers do not need non-local information about a function's call sites to infer the function's return type. Generally, this bottom-up approach provides programmers with a clear intuition about the flow of type information.

However, in some limited contexts, inference proceeds "top-down" from the context of an expression. Where this happens, it is called contextual typing. Contextual typing helps tools provide excellent information when a programmer is using a type but may not know all of the details of the type. For example, in the jQuery example, above, the programmer supplies a function expression as the second parameter to the 'get' method. During typing of that expression, tools can assume that the type of the function expression is as given in the 'get' signature and can provide a template that includes parameter names and types.

```
$.get("http://mysite.org/divContent",
      function (data) {
          $("div").text(data);  // TypeScript infers data is a string
      }
);
```