For this we employ some open source from a browser codebase. This chapter takes this part of the process as a black box, so let's now assume that after receiving the FBML and sending it through this flow, we will have a tree-like structure called FBMLNode, which gives us the ability to query the tag, attribute key-value pairs, and raw content at any node in the generated syntax tree, and recursively query child elements.

Jumping to the highest level, note that FBML appears all over the Facebook site: application "canvas" pages, the content of News Feed stories, and the content of profile boxes, to name a few places. Each of these contexts or "flavors" of FBML defines constraints on the input; for instance, canvas pages allow iframes, whereas profile boxes do not. And naturally, because FBML maintains privacy of data in a way similar to the API, the execution context must include both the viewing user and the application ID of the application generating the content.

So before we actually engage with the payload of FBML, we start with the rules of our environment, encompassed in the FBMLFlavor class in Example 6-24.

*EXAMPLE 6-24. The FBMLFlavor class*

```
abstract class FBMLFlavor {

// constructor takes array containing user and application_id
  public function FBMLFlavor ($environment_array) { ... }
  public function check($category) {
    $method_name = 'allows_' . $category;
    if (method_exists($this,$method_name)) {
      $category_allowed = $this->$method_name();
    } else {
      $category_allowed = $this->_default();
    }
    if (!$category_allowed))
      throw new FBMLException('Forbidden tag category '.$category.' in this flavor.');
  }
  protected abstract function _default();
}
```

The flow instantiates a child of this abstract flavor class that corresponds to the page or element rendering the FBML. Example 6-25 shows an example.

*EXAMPLE 6-25. An instantiation of the FBMLFlavor class*

```
class ProfileBoxFBMLFlavor extends FBMLFlavor {
  protected function _default() { return true; }
  public function allows_redirect() { return false; }
  public function allows_iframes() { return false; }
  public allows_visible_to() { return $this->_default(); }
 // ...
}
```

The flavor's design is simple: it contains the privacy context (user and application) and implements the check method, setting up the rules for the meaty logic contained in the FBMLImplementation class shown later. Much like the Platform API's implementation layer, the