```
    return ''; // unknown value? not visible

  $viewer = $this->get_env('user');
  $predicate = can_see($viewer, $uid, 'user', $what);
  return $this->render_if($node, $predicate); // handles the else case for us
}

// helper for the fb_if family of functions
protected function render_if($node, $predicate) {
  if ($predicate) {
    return $this->render_children($node);
  } else {
    return $this->render_else($node);
  }
}

protected function render_else($node) {
  $html = '';
  foreach ($node->get_children() as $child) {
    if ($child->get_tag_name() == 'fb:else') {
      $html .= $child->render_children($this);
    }
  }

  return $html;
}

public function fb_else($ignored_node) { return ''; }
```

If the can_see check passes for the specified viewer-object pair, the engine renders the children of the <fb:if-can-see> node recursively. Otherwise, the content below any optional <fb:else> children is rendered. Notice how fb_if_can_see directly accesses the <fb:else> children; if <fb:else> appears outside one of these "if-style" FBML tags, the tag and its children return no content at all. So FBML is not just a simple swap routine; it is aware of the structure of the document, and thus can incorporate elements of conditional flow.

Putting it all together

Each of the functions just discussed needs to be registered as a callback that is used while parsing the input FBML. At Facebook (and in the open source Platform implementation), this "black box" parser is written in C as an extension to PHP, and each of these callbacks lives in the PHP tree itself. To complete the high-level flow, we must declare these tags to the FBML parsing engine. As elsewhere, Example 6-29 is highly edited for simplicity.

EXAMPLE 6-29. The FBML main evaluation flow

```
// As input to this flow:
// $fbml_impl - the implementation instantiated above
// $fbml_from_callback - the raw FBML string created by the external application

// a list of "Direct HTML" tags
$html_special = $fbml_impl->get_special_html_tags();
```