outlined in the next section to effectively build web services of any kind, regardless of whether the data in the developer's storehouse is public or private.

But note that Facebook users do not consider their Facebook data to be fully public. So after our technical overview, we'll look at maintaining Facebook-level privacy through the main authentication means in the Platform API: user sessions.

Data: Creating an XML Web Service

In order to add basic social context to an example application, we've established the existence of two remote method calls, `friends.get` and `users.getInfo`. The internal functions accessing this data are likely sitting in a library somewhere in the Facebook code tree, serving similar requests on the Facebook site. Example 6-6 shows some examples.

EXAMPLE 6-6. Example social data mappings

```
function friends_get($session_user) { ... }
function users_getInfo($session_user, $input_users, $input_fields) { ... }
```

We now build a simple web service, transforming GET and POST input over HTTP to internal stack calls, and outputting the results as XML. In the Facebook Platform's case, the name of the destination method and its arguments are passed in the HTTP request, as well as some credentials specific to the calling application (an assigned "api key"), specific to a user-application pair (a "user session key"), and specific to the request instance itself (a request "signature"). We'll address the session key later in "A Simple Web Service Authentication Handshake." The high-level sequence for servicing a request to *http://api.facebook.com* is then:

1. Examine passed credentials ("A Simple Web Service Authentication Handshake") to verify the invoking application's identity, user's current authorization of the application, and authenticity of the request.
2. Interpret the incoming GET/POST as a method call with reasonable arguments.
3. Dispatch a single call to internal method and collect the result as in-memory data structures.
4. Transform these structures into a known output form (e.g., XML or JSON) and return.

The main points of difficulty in constructing interfaces consumed externally usually arise in steps 2 and 4. Consistently maintaining, synchronizing, and documenting the data interfaces for an external consumer is important, and constructing the skeleton code to ensure this consistency by hand is a thankless and time-consuming job. Additionally, we may need to make this data available to internal services written in many languages, or communicate results to an external developer in different web protocols such as XML, JSON, or SOAP.

Here, then, the beautiful solution is the use of metadata to encapsulate the types and signatures describing the APIs. Engineers at Facebook have created an open source cross-language

Download at WoweBook.Com