XML-RPC and SOAP Programming Guide

Developer

Contents

Introduction to XML-RPC and SOAP Programming Guide 4

Who Should Read This Document 4

Organization of This Document 5

About AppleScript's Support for XML-RPC and SOAP 6

The XML-RPC and SOAP Protocols 6

AppleScript Support for Remote Procedure Calls 7

- Remote Procedure Calls From Scripts 9
- Remote Procedure Calls From Applications 13

Making Remote Procedure Calls From Scripts 18

Scripting an XML-RPC Call 18

Scripting a SOAP Request 22

- A Simple Translation Script 22
- A More Complex Translation Script 24

Making Remote Procedure Calls From Applications 27

Making an XML-RPC Call 27

- Setting Up a Project 27
- Includes, Constants, and Declarations 28
- A Main Function That Makes an XML-RPC Call 29
- Displaying XML-RPC Debug Information 32

Making a SOAP Request 35

- Setting Up a Project 36
- Includes, Constants, and Declarations 36
- A Main Function That Makes a SOAP Request 37
- Creating Parameter List Records 41
- Displaying SOAP Request Debug Information 42

Document Revision History 45

Figures and Listings

About AppleScript's Support for XML-RPC and SOAP 6

Figure 1-1 The Apple Event Manager processing a remote procedure call Apple Event. 8
Listing 1-1 Sample list of misspelled words returned by spellCheck remote procedure call. 11
Listing 1-2 Event class and event ID constants for remote procedure call Apple events. 14
Listing 1-3 Address descriptor type for remote procedure call Apple events. 14
Listing 1-4 Constants for specifying a SOAP schema. 15
Listing 1-5 Constants used in constructing an XML-RPC or SOAP request in a remote procedure call Apple event. 15
Listing 1-6 Constants for turning on the Apple Event Manager's remote procedure call debugging. 16
Listing 1-7 Key constants for remote procedure call debugging attributes. 16

Making Remote Procedure Calls From Scripts 18

Listing 2-1 A script that makes an XML-RPC request to check the spelling of a text phrase. 18
Listing 2-2 A simple script that calls a SOAP translation server. 22
Listing 2-3 A more detailed script that calls a SOAP translation server. 24

Making Remote Procedure Calls From Applications 27

Listing 3-1 Includes, constants, and declarations for making an XML-RPC call. 28
Listing 3-2 main function to send an XML-RPC Apple event. 30
Listing 3-3 A function to display debug information from an XML-RPC reply Apple event. 33
Listing 3-4 Sample header and data from an XML-RPC post and the reply. 34
Listing 3-5 Includes, constants, and declarations for making a SOAP request. 36
Listing 3-6 main function to send a SOAP Apple event. 38
Listing 3-7 A function to create a parameter list record for a SOAP request Apple event. 41
Listing 3-8 Sample header and data from a SOAP post and the reply. 42

Introduction to XML-RPC and SOAP Programming Guide

Note: This document was previously titled "XML-RPC and SOAP Support."

XML-RPC and SOAP Programming Guide describes how to use Apple Script and the Apple Event Manager in OS X to make remote procedure calls using the XML-RPC and SOAP (Simple Object Access Protocol) protocols.

XML-RPC is a protocol for using XML and HTTP to make remote procedure calls over the Internet. **SOAP (Simple Object Access Protocol)** is a remote procedure call protocol designed for exchanging information in a distributed environment, where a server may consist of a hierarchy of objects.

This book describes only how to send XML-RPC and SOAP requests, not how to serve them.

Note: Web service protocols such as XML-RPC and SOAP are evolving, as is the support for them in OS X. The scripts and code samples in this document depend on third-party web services that are also subject to change. Though the samples have been tested as published, your experience may vary.

The sample code in this book can be adapted for Carbon applications, Cocoa applications, simple tools, or other code. However, there is no specific Cocoa class support provided.

Who Should Read This Document

To take full advantage of this document, you should be familiar with AppleScript, either through writing AppleScript scripts or creating scriptable applications. You can learn more about these topics in AppleScript Documentation. In particular, see *AppleScript Overview* and *Apple Events Programming Guide*.

You should also be familiar with the XML-RPC and SOAP protocols. You can find information on these protocols at third-party websites. For example, the XML-RPC specification is currently described at http://www.xmlrpc.com/spec and the SOAP specification at http://www.w3.org/TR/.

Organization of This Document

This document is organized into the following chapters:

- "About AppleScript's Support for XML-RPC and SOAP" (page 6) provides a brief introduction to the XML-RPC and SOAP protocols, then describes the scripting support in OS X version 10.1 (and later) for these protocols, including the syntax for script statements and the APIs for making remote procedure calls from applications or other code.
- "Making Remote Procedure Calls From Scripts" (page 18) provides sample scripts and step by step descriptions for making XML-RPC and SOAP requests from scripts.
- "Making Remote Procedure Calls From Applications" (page 27) provides sample code and step by step descriptions for making XML-RPC and SOAP requests from applications and other code.
- "Document Revision History" (page 45) describes the history of this book.

About AppleScript's Support for XML-RPC and SOAP

Starting with OS X version 10.1, AppleScript and the Apple Event Manager provide XML-RPC and SOAP support such that:

- Scripters can make XML-RPC calls and SOAP requests from scripts.
- Developers can make XML-RPC calls and SOAP requests from applications or other code by sending Apple events.

The XML-RPC and SOAP Protocols

A **remote procedure call** is a request to a server application at another location to perform operations and return information. **XML-RPC** is a simple protocol that allows software running in different environments to make remote procedure calls over the Internet. XML-RPC uses two industry standards: XML (extensible markup language) for encoding messages, and HTTP (hypertext transfer protocol) for transporting them. A properly formatted XML-RPC message is an HTTP POST request whose body is in XML. The specified remote server executes the requested call and returns any requested data in XML format.

XML-RPC recognizes procedure parameters by position. Parameters and return values can be simple types such as numbers, strings, and dates, or more complex types such as structures and arrays. To learn more about XML-RPC messages, see the XML-RPC specification at http://www.xmlrpc.com/spec.

SOAP (Simple Object Access Protocol) is an RPC protocol designed for a distributed environment, where a server may consist of a hierarchy of objects whose methods can be called over the Internet. A goal of SOAP is to establish a standard protocol that will serve both web service providers and service users. As with other remote procedure call protocols, SOAP uses XML to encode messages and HTTP to transport them. A SOAP request contains a header and an envelope; the envelope in turn contains the body of the request.

One key difference between the SOAP and XML-RPC protocols is that with SOAP, parameters are notational (a request must encode the method parameter names within its XML), rather than positional (recognized by position). To learn more about SOAP messages, see the SOAP specification at http://www.w3.org/TR/.

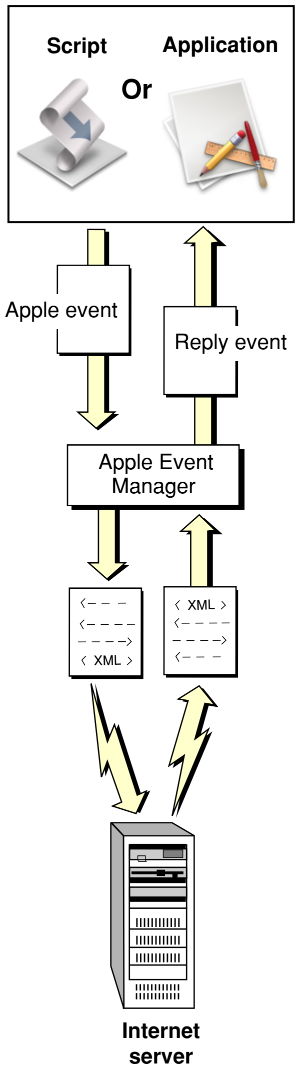
Remote procedure calls provide a powerful tool for accessing services over the Internet. For example, there are already a variety of web-based servers that can check spelling, translate text between languages, provide stock prices, supply weather and traffic information, and more. You can find available services at sites such as XMethods at http://www.xmethods.net/. There you can also find information you'll need to make remote procedure calls to these services.

AppleScript Support for Remote Procedure Calls

This section describes how to make XML-RPC and SOAP requests from scripts and from applications or other code. Using this support, both scripters and developers can take advantage of the growing number of XML-RPC and SOAP servers available on the Internet.

With OS X version 10.1, the Apple Event Manager is able to process Apple events that encapsulate remote procedure calls, whether the events originate from scripts or applications. Figure 2-1 (page 8) shows a simplified view of this process.

Figure 1-1 The Apple Event Manager processing a remote procedure call Apple Event.



The Apple Event Manager recognizes a remote procedure call Apple event by its address descriptor, which uses the descriptor type `typeApplicationURL`. To process a remote procedure call Apple event (assuming the event and the XML it contains are properly formatted), the Apple Event Manager performs these basic steps:

1. It uses information from the Apple event to build the XML that represents the remote procedure call.
2. It opens a connection to the specified server.
3. It sends the XML via HTTP POST request.

4. It waits for a response.
5. It parses the XML of the response.
6. It constructs a reply Apple event.
7. It returns the reply.

Scripters can take advantage of this capability by adding statements to their scripts that specify XML-RPC or SOAP requests. Application developers can create and send Apple events directly to invoke XML-RPC or SOAP requests.

Remote Procedure Calls From Scripts

When you compile and execute a script, the AppleScript component works with the Apple Event Manager to convert script statements into Apple events that are sent to the specified applications. The following sections describe the terminology and syntax for making XML-RPC calls and SOAP requests from scripts.

AppleScript Terminology for Remote Procedure Calls

AppleScript 1.7 for OS X version 10.1 can specify Internet applications as targets of Tell statements or within Using Terms From blocks. When you specify an XML-RPC server or a SOAP server in one of these statements, the Apple Event Manager generates a dictionary that makes the terminology for remote procedure calls available to your script. That terminology includes the following terms that specify XML-RPC and SOAP requests:

- `call xmlrpc`
- `call soap`

The next sections describe how to use AppleScript's remote procedure call terminology.

XML-RPC Script Statements

To make an XML-RPC request from a script, you must specify an XML-RPC server as the target application for the request. To do so, you use a statement like the following, which specifies an XML-RPC server that provides spell-checking services over the Internet:

```
tell application "http://www.stuffeddog.com/speller/speller-rpc.cgi"
```

In this statement, the XML-RPC server application is `speller-rpc.cgi` and the remote location is specified by the URL `http://www.stuffeddog.com/speller/`. You can also use the following alternate syntax (where the symbol ¬ (Option-l) denotes a continuation of one script statement onto more than one line), but this form is converted to the above form when the script is compiled:

```
tell application "speller-rpc.cgi" of machine ¬
     "http://www.stuffeddog.com/speller"
```

Once you have specified an XML-RPC server, you can make remote procedure calls within a script block (such as a Tell statement) that specifies that server:

```
tell application "http://www.stuffeddog.com/speller/speller-rpc.cgi"
    -- remote procedure calls here
end tell
```

To specify a name and parameters for the remote procedure call, you use script statements like the following:

```
set returnValue to¬
    call xmlrpc {method name:"someMethod",¬
        parameters: {parameter1, parameter2} }
```

This statement specifies a remote procedure call with two parameters. (You use the same terminology, `method name:`, whether specifying a function or method.) The parameters are represented as a list. The returned information is stored in the variable `returnValue`. Because XML-RPC parameter syntax is positional, you don't need to provide the parameter names for a given procedure call. An XML-RPC server returns an error if the passed XML is not properly formatted.

The following example combines a Tell block with a remote procedure call to the spell-checking server specified earlier.

```
set myText to "My frst naem is John"
tell application "http://www.stuffeddog.com/speller/speller-rpc.cgi"
    set returnValue to call xmlrpc {method name:"speller.spellCheck",¬
        parameters: {myText} }
end tell
```

This sample shows how easy it can be to set up a remote procedure call from a script. The first line sets a local variable to a text string to be checked. Then the Tell statement specifies an Internet spell-checking server. The single statement in the Tell block calls the `speller.spellCheck` function, passing the local string variable as the single parameter (the text to be checked) and storing the result in a second local variable. This particular remote procedure returns a list that contains, for each misspelled word, a list of suggested corrections, the location of the word in the text, and the misspelled word itself. Listing 2-1 (page 11) shows an example of such a list.

Listing 1-1 Sample list of misspelled words returned by spellCheck remote procedure call.

```
{{suggestions:{"fast", "fest", "first", "fist", "Forst",
"frat", "fret", "frist", "frit", "frost", "frot", "fust"},
location:4, |word|:"frst"}, {suggestions:{"haem", "na em",
"na-em", "naam", "nae", "nae m", "nae-m", "nael", "Naim",
"nam", "name", "neem"}, location:9, |word|:"naem"}}
```

Note that this is not the raw data returned from the remote procedure call. When you execute the script shown above, the AppleScript component calls on the Apple Event Manager to convert the `call xmlrpc` statement to an Apple event and send the event to the specified server. The Apple Event Manager recognizes the event and processes it by formatting the remote procedure call into proper XML, opening a connection, sending the message, waiting for a reply, formatting the returned XML into an Apple event, and returning the event.

For a complete script that uses remote procedure calls to check spelling, see "Scripting an XML-RPC Call" (page 18).

SOAP Script Statements

The process for making SOAP requests in AppleScript scripts is very similar to the process for XML-RPC, though it uses the term `call soap` rather than `call xmlrpc`. To specify a SOAP server as the target application for a SOAP request, you use a statement like the following:

```
tell application "http://services.xmethods.net:80/perl/soaplite.cgi"
```

In this statement, the SOAP server application is `soaplite.cgi`, a server that can perform English to French translation, and the remote location is specified by the URL `http://services.xmethods.net:80/perl/`. Once you have specified a SOAP server on a remote machine, you can make SOAP requests within a script block that specifies that server:

```
tell application "http://services.xmethods.net:80/perl/soaplite.cgi"
    -- SOAP requests here
end tell
```

To specify a method name and parameters for the SOAP request, you use script statements like the following (where the symbol ¬ (Option-l) denotes a continuation of one script statement onto more than one line):

```
    set returnValue to call soap {method name:"BabelFish", ¬
        method namespace uri:"urn:xmethodsBabelFish", ¬
        parameters:{translationmode:direction as string, ¬
        sourcedata:theText as string}, ¬
        SOAPAction:"urn:xmethodsBabelFish#BabelFish"}
```

This statement specifies a SOAP request to a method named `BabelFish` with two parameters, `translationmode` (such as English to French) and `sourcedata` (the text to be translated). The returned information (translated text) is stored in the variable `returnValue`. Unlike the case with XML-RPC calls, for a SOAP request you must specify the names of the parameters for the specified method. While an XML-RPC call represents parameters as a list, a SOAP request represents them as a record. The server will return an error if the passed XML is not properly formatted.

The following example combines a Tell block with a SOAP request to the translation server specified earlier.

```
set theText to "The spirit is willing but the flesh is weak."
set direction to "en_fr"
tell application "http://services.xmethods.net:80/perl/soaplite.cgi"
    set resultText to call soap {method name:"BabelFish", ¬
        method namespace uri:"urn:xmethodsBabelFish", ¬
        parameters:{translationmode:direction as string, ¬
        sourcedata:theText as string}, ¬
        SOAPAction:"urn:xmethodsBabelFish#BabelFish"}
end tell
```

The first two lines set local variables for the text to be translated and the direction of translation (from English to French). The Tell statement specifies an Internet language translation server. The `call soap` statement in the Tell block calls the `BabelFish` function, specifying two parameters by name (`translationmode` and `sourcedata`), and passing the local variables as the parameter values. All the terms shown in the `call soap` statement in this example are required except `parameters:` (because a SOAP method may have no parameters). The resulting translated text is stored in another local variable.

You obtain values for the `method namespace uri:` and `SOAPAction:` terms from the services themselves. For example, many SOAP services publish the information needed to use their services at sites such as XMethods at http://www.xmethods.net/. You can learn more about SOAP name spaces and SOAPAction in the SOAP specification at http://www.w3.org/TR/.

When you execute this script, AppleScript and the Apple Event Manager take care of formatting the SOAP request into proper XML, opening a connection, sending the message, waiting for a reply, formatting the returned XML into an Apple event, and returning the event.

For a complete script that uses SOAP requests to translate English text to French, see "Scripting a SOAP Request" (page 22).

Remote Procedure Calls From Applications

As described earlier, the Apple Event Manager is now able (in OS X version 10.1) to process Apple events that encapsulate remote procedure calls, whether the events originate from scripts or applications. Assuming the Apple event and the XML it contains are properly formatted, the Apple Event Manager extracts the XML, establishes a connection to the specified server, sends the request via HTTP, waits for a response, parses the XML of the response, constructs a reply Apple event, and returns the reply.

To take advantage of this support, applications and other code can create Apple events to send either XML-RPC or Soap requests. The process includes the following steps:

1. Link with `Carbon.framework`.
2. Prepare any necessary Apple event descriptors, including an address descriptor of type `typeApplicationURL` that specifies the target for the request (a remote server).
3. Create an Apple event with event class `kAERPCClass` and event type `kAEXMLRPCScheme` (for XML-RPC) or `kAESOAPScheme` (for SOAP), and with the target address descriptor created previously.
4. Create the direct object for the Apple event and insert parameters for the method name, method parameter list, and any other required information for the remote procedure call.
5. Insert the direct object into the Apple event.
6. Send the Apple event with `AESend`.
7. Process the reply Apple event to extract any needed information.

More detailed steps for sending an XML-RPC Apple event are shown in "Sending an XML-RPC Apple Event" (page 16) and for a SOAP request in "Sending a SOAP Apple Event" (page 17). For examples with sample code, see "Making Remote Procedure Calls From Applications" (page 27).

There is some overhead in creating Apple events to send remote procedure calls and in extracting information from the reply Apple event, but you gain the advantage of having the Apple Event Manager build the required XML for you. If, however, you already have code that, for example, automatically generates XML for remote procedure calls, it may be more efficient for you to write your own code to make the remote procedure calls as well.

Apple Event Manager API for Remote Procedure Calls

This section describes some of the key constants used to construct remote procedure call Apple events. These constants are defined in `AEDataModel.h` (in `AE.framework`, a subframework of `ApplicationServices.framework`). For step-by-step instructions that show how to create Apple events using these constants, see "Making Remote Procedure Calls From Applications" (page 27).

The Apple Event Manager defines one event class constant and two event ID constants for remote procedure call Apple events. These constants are shown in Listing 2-2 (page 14). You use the constant `kAERPCClass` as the event class for both XML-RPC and SOAP requests. To specify the request type, you use either `kAEXMLRPCScheme` or `kAESOAPScheme` for the event ID.

Listing 1-2 Event class and event ID constants for remote procedure call Apple events.

```
    kAERPCClass     = 'rpc ', /* for outgoing XML events */
    kAEXMLRPCScheme = 'RPC2', /* event ID: send to XMLRPC endpoint */
    kAESOAPScheme   = 'SOAP', /* event ID: send to SOAP endpoint */
```

Listing 2-3 (page 14) shows the constant `typeApplicationURL`. An Apple event for a remote procedure call must have an address descriptor of this type that specifies the target for the request. The Apple Event Manager recognizes this type as a remote procedure call and processes it appropriately.

Listing 1-3 Address descriptor type for remote procedure call Apple events.

```
enum {
    //...some constants not shown
    typeApplicationURL = 'aprl',
    //...
};
```

The SOAP specification defines a schema for encoding (or serializing) information. The Apple Event Manager can work with both the 1999 (or SOAP specification version 1.1) and 2001 (SOAP specification version 1.2) schemas. Listing 2-4 (page 15) shows the constants for specifying the SOAP schema used to format the SOAP request in a remote procedure call Apple event.

You can specify a serialization schema by adding a parameter of type `typeType` and key `keySOAPSchemaVersion` to the direct object of a SOAP request Apple event. If you do not specify a schema, the default is `kSOAP1999Schema`.

Listing 1-4 Constants for specifying a SOAP schema.

```
enum {
    kSOAP1999Schema = 'ss99',
    kSOAP2001Schema = 'ss01',
    //...
    keySOAPSchemaVersion = 'ssch'
};
```

Listing 2-5 (page 15) shows some of the constants you use to identify the components of an XML-RPC or SOAP request. When you create a remote procedure call Apple event, you use these constants to add various information about the request to the direct object of the Apple event:

- You use the key `keyRPCMethodName` to add an Apple event parameter that specifies the procedure or method name for the request.
- After you build a descriptor list that describes the parameters for a remote procedure call, you insert it into the direct object for the Apple event using the key `keyRPCMethodParam`.
- For a SOAP request, you add the required SOAPAction header to the direct object using the key `keySOAPAction`.
- For a SOAP request, you add the required SOAP name space URI to the direct object using the key `keySOAPMethodNameSpaceURI`.

Listing 1-5 Constants used in constructing an XML-RPC or SOAP request in a remote procedure call Apple event.

```
    keyRPCMethodName    = 'meth', /* name of the method to call */
    keyRPCMethodParam   = 'parm', /* the list (or structure) of parameters*/
    keySOAPAction       = 'sact', /* the SOAPAction header */
    keySOAPMethodNameSpaceURI= 'mspu',/* Required namespace URI */
```

Listing 2-6 (page 16) shows constants you can specify in an attribute to a remote procedure call Apple event to turn on Apple Event Manager debugging. Depending on which debug information you specify with these constants, the reply Apple event from an XML-RPC or SOAP request can supply the header or the body of the outgoing request, or of the reply.

Listing 1-6 Constants for turning on the Apple Event Manager's remote procedure call debugging.

```
enum {
    kAEDebugPOSTHeader = (1 << 0), /* headers of the HTTP post - typeChar */
    kAEDebugReplyHeader = (1 << 1), /* headers returned by the server */
    kAEDebugXMLRequest = (1 << 2), /* the XML request we sent */
    kAEDebugXMLResponse = (1 << 3), /* the XML reply from the server */
    kAEDebugXMLDebugAll = 0xffffffff/* everything! */
};
```

Depending on which debugging flags you specify with the constants shown in Listing 2-6 (page 16), one or more attributes is added to the Apple event. You can extract those attributes using the keys shown in Listing 2-7 (page 16).

Listing 1-7 Key constants for remote procedure call debugging attributes.

```
    keyAEPOSTHeaderData = 'phed', /* header of request to the server */
    keyAEReplyHeaderData = 'rhed', /* header of server reply */
    keyAEXMLRequestData = 'xreq', /* body of request to the server */
    keyAEXMLReplyData = 'xrep', /* body of server reply */
```

Sending an XML-RPC Apple Event

To make an XML-RPC request from your application or other code, you'll need to create an Apple event that encapsulates the procedure call and call `AESend` to send it. This section describes the steps you need to follow. To see those steps implemented in code, see "Making an XML-RPC Call" (page 27).

To use an Apple event to execute an XML-RPC request, you do the following:

1. Link with `Carbon.framework`.
2. Create a target address descriptor of type `typeApplicationURL` that specifies the target for the request (a remote server).
3. Create an Apple event with event class `kAERPCClass`, event ID `KAEXMLRPCScheme`, and with the target address descriptor created in a previous step.

4. Create the direct object for the Apple event.
5. Insert the method name, using the key `keyRPCMethodName`.
6. Create a list descriptor for the remote procedure call parameters and insert each parameter (using the key `keyRPCMethodParam`).
7. Add the parameter list to the direct object.
8. Insert the direct object into the Apple event.
9. Optionally turn on debugging by adding an attribute to the event with key `keyXMLDebuggingAttr`.
10. Send the Apple event with `AESend`.
11. Process the reply Apple event to extract any needed information. If you turned on debugging, you can extract debugging information, according to which of the debugging constants (described above) you specified. For example, you can examine the header or the body of the posted message, or of the reply.

Sending a SOAP Apple Event

To make a SOAP request from your application or other code, you'll need to create an Apple event that encapsulates the request and call `AESend` to send it. This section describes the steps you need to follow. To see those steps implemented in code, see "Making a SOAP Request" (page 35).

To use an Apple event to execute a SOAP request, you do the following:

1. Link with `Carbon.framework`.
2. Create a target address descriptor of type `typeApplicationURL` that specifies the target for the request (a remote server).
3. Create an Apple event with event class `kAERPCClass`, event ID `kAESOAPScheme`, and with the target address descriptor created in a previous step.
4. Create the direct object for the Apple event.
5. Insert the method name, using the key `keyRPCMethodName`.
6. Create a list descriptor for the SOAP method parameters and insert each parameter as character data.
7. Add the parameter list to the direct object, using the key `keyRPCMethodParam`.
8. Add the SOAP name space URI to the direct object, using the key `keySOAPMethodNameSpaceURI`.
9. Insert the direct object into the Apple event.
10. Optionally turn on debugging by adding an attribute to the event with key `keyXMLDebuggingAttr`.
11. Send the Apple event with `AESend`.
12. Process the reply Apple event to extract any needed information. If you turned on debugging, you can extract debugging information, according to which of the constants described above you specified. For example, you may be able to examine the header or the body of the posted message, or of the reply.

Making Remote Procedure Calls From Scripts

Starting with OS X version 10.1, the Apple Event Manager provides support for using the XML-RPC and SOAP protocols to make remote procedure calls from AppleScript scripts and from applications. This chapter provides sample scripts that show how to make remote procedure calls from scripts.

This chapter assumes you are familiar with the material in "Introduction to XML-RPC and SOAP Programming Guide" (page 4). To test any of the scripts shown in this chapter, you must have an Internet connection.

Warning: The script examples in this chapter may not work because the web services they rely on may no longer be available. You can find additional script samples that make use of web services at `/Library/Scripts/Internet Services`. For example, the `Stock Quote` script uses a web service to look up the price for a stock specified by its symbol, while the `Current Temperature by Zipcode` script looks up the temperature for a specified Zip code.

Scripting an XML-RPC Call

To make an XML-RPC request from a script, you use the AppleScript term `call xmlrpc`. The syntax for this term is described in "XML-RPC Script Statements" (page 9). You can find available XML-RPC services at sites such as XMethods at http://www.xmethods.net/. There you can also find information about the parameters and return values for remote procedure calls to these services.

Listing 3-1 (page 18) shows a script that prompts a user for text, makes an XML-RPC request to an Internet server to check the spelling for that text, prompts the user to correct any words that may have been misspelled, and displays the final text.

Listing 2-1 A script that makes an XML-RPC request to check the spelling of a text phrase.

```
---- Main script -----------------------------------------
-- Supply default text to spell check.
set spellCheckText to "My frst naem is John"

----Query user for different text to check.
set dialogResult to display dialog ¬
```

```
    "Enter a phrase to spell check:" default answer spellCheckText

if button returned of dialogResult is "OK" then
    set spellCheckText to text returned of dialogResult
    -- Call spellCheck handler.
    set resultList to spellCheck(spellCheckText)
    (*
    The returned data looks something like this:
        {{suggestions:{"fast", "fest", "first", "fist", "Forst",
        "frat", "fret", "frist", "frit", "frost", "frot", "fust"},
        location:4, |word|:"frst"}, {suggestions:{"haem", "na em",
        "na-em", "naam", "nae", "nae m", "nae-m", "nael", "Naim",
        "nam", "name", "neem"}, location:9, |word|:"naem"}}
    *)
    -- Make list of words to spellcheck.
    set wordList to every word of spellCheckText

    -- Give user chance to correct each misspelled word.
    repeat with eachItem in resultList
        set newWord to choose from list suggestions of eachItem ¬
            with prompt "You misspelled \"" & |word| of eachItem & ¬
            "\"" without multiple selections allowed

        -- If user selected a corrected word, insert it into list
        if (newWord as string) is not equal to "false" then
            set wordIndex to ¬
                findIt(every word of spellCheckText, location of eachItem)
            copy newWord to item wordIndex of wordList
        end if
    end repeat

    -- Display corrected text.
    set spellCheckText to ""
    repeat with oneWord in wordList
        set spellCheckText to spellCheckText & oneWord & " "
    end repeat

    display dialog "Corrected text: " & spellCheckText
end if

-- spellCheck handler ------------------------------------------
-- This handler makes the remote procedure call.
on spellCheck(sentence)
    tell application "http://www.stuffeddog.com/speller/speller-rpc.cgi"
        return call xmlrpc {method name:"speller.spellCheck", ¬
        parameters:sentence}
    end tell
end spellCheck



-- findIt handler ------------------------------------
-- The "textList" parameter is a list of the words in the original text.
-- The "index" parameter is the character index of a misspelled word.
-- This handler returns the word at that index.
-- For example, the misspelled word at character index four is "frst".
--      Its word index is 2 (the 2nd word in the original text).
on findIt(textList, index)
    set curLength to 0
    set ixWord to 1

    repeat with oneWord in textList
        set curLength to curLength + (length of oneWord) + 1
        if curLength ≥ index then
            exit repeat
        end if
        set ixWord to ixWord + 1
    end repeat
    log ixWord
    return ixWord
```

```
end findIt
```

This script has a main section and two handlers. In the main section, it performs the following actions:

1. It sets a default text string to be checked.
2. It displays a dialog that prompts the user to enter different text.
3. If the user accepts the dialog, it calls the `spellCheck` handler to check the spelling. That handler, described below, contains the only script statements needed to make a remote procedure call to a spell-checking server.

   The handler returns a list that contains, for each misspelled word, a list of suggested corrections, the character location of the word in the text, and the misspelled word itself. The returned list looks something like this:

   ```
   The returned data looks something like this:
           {{suggestions:{"fast", "fest", "first", "fist", "Forst",
           "frat", "fret", "frist", "frit", "frost", "frot", "fust"},
           location:4, |word|:"frst"}, {suggestions:{"haem", "na em",
           "na-em", "naam", "nae", "nae m", "nae-m", "nael", "Naim",
           "nam", "name", "neem"}, location:9, |word|:"naem"}}
   ```

   Note that this is not the raw data returned from the remote procedure call. Rather, the Apple Event Manager has interpreted the XML returned by the procedure call and built an Apple event to encapsulate it. This list of records is the result. Because `word` is a reserved word in AppleScript, it is enclosed in vertical bars (`|word|`) when used as an identifier (in this case as a label in a record).
4. For each word (if any) in the returned list of misspelled words, it lets the user choose a correction (using the standard scripting addition `choose from list`).

   It then calls the `findIt` handler to find the location of the misspelled word in the text phrase and replaces it with the user choice.
5. It displays the corrected text (possibly the same as the original text, if no corrections were made).

The `spellCheck` handler contains the one and only statement in the script that makes a remote procedure call. It performs the following operations:

1. It uses a Tell statement to identify the location of the remote XML-RPC server (`http://www.stuffeddog.com/speller/speller-rpc.cgi`).
2. It uses the AppleScript term `call xmlrpc` to make the remote procedure call, specifying the method name (`speller.spellCheck`) and passing the specified text for the single parameter.

3. It returns the result of the remote procedure call. That consists of a list that contains, for each misspelled word, a list of suggested corrections, the location of the word in the text, and the misspelled word itself.

Note that AppleScript, working through the Apple Event Manager, did all the work of formatting the `call xmlrpc` script statement into proper XML, opening a connection to the specified server, sending the message, waiting for a reply, formatting the returned XML into an Apple event, and returning the event.

The `findIt` handler merely returns the word position in the original text of the word that corresponds to the passed character index of a misspelled word (returned by the remote procedure handler). For example, the character index of the first misspelled word (`frst`) is 4. That word is the second word in the original text, so `findit` would return the value 2.

For a script that shows a more flexible type of handler and includes error handling, see Listing 3-3 (page 24).

Scripting a SOAP Request

To make a SOAP request from a script, you use the AppleScript term `call soap`. The syntax for these calls is described in "SOAP Script Statements" (page 11). You can find available SOAP services at sites such as XMethods at http://www.xmethods.net/. There you can also find information about the parameters and return values for SOAP requests to these services.

A Simple Translation Script

Listing 3-2 (page 22) shows a script that prompts a user for text, makes a SOAP request to an Internet server to translate that text to French, and displays the translated text.

Listing 2-2 A simple script that calls a SOAP translation server.

```
---- Main script ----------------------------------------
-- Supply default text to translate.
set defaultText to "The spirit is willing but the mind is weak"

--Display dialog and let user type different text to translate.
display dialog "Enter the text to translate into French:" ¬
    default answer defaultText
set this_text to the text returned of the result

-- Call translation handler
```

```
set the new_text to translate("en_fr", this_text)

--Show translated text and allow user to copy it to Clipboard.
display dialog new_text buttons {"Clipboard", "OK"} default button 2
if the button returned of the result is "Clipboard" then
    set the clipboard to new_text
end if

-- translate handler ------------------------------------------
-- This handler makes the SOAP request.
on translate(direction, theText)
    tell application "http://services.xmethods.net:80/perl/soaplite.cgi"
        return call soap {method name:"BabelFish", ¬
        method namespace uri:"urn:xmethodsBabelFish", ¬
        parameters:{translationmode:direction as string, ¬
        sourcedata:theText as string}, ¬
        SOAPAction:"urn:xmethodsBabelFish#BabelFish"}
    end tell
end translate
```

This script starts has a main section and one handler to make the translation SOAP request. In the main section, it performs the following actions:

1. It sets a default text string to be translated from English to French.
2. It displays a dialog that prompts the user to enter different text.
3. It calls the `translate` handler to translate the text. That handler, described below, contains the only script statements needed to make a SOAP request to a translation server.

   The handler returns the translated text.
4. It displays the translated text and allows the user to copy it to the Clipboard.

The `translate` handler contains the one and only statement in the script that makes a SOAP request. It performs the following operations:

1. It uses a Tell statement to identify the location of the remote SOAP server (`http://services.xmethods.net:80/perl/soaplite.cgi`).

2. It uses the AppleScript term `call soap` to make the SOAP request. You can obtain certain values you need to make a SOAP request from the service itself. In this example, the call includes the following:
   - `method name:"BabelFish"` specifies the required method name
   - `method namespace uri:"urn:xmethodsBabelFish"` specifies the required method namespace URI
   - `parameters:{translationmode:direction as string, sourcedata:theText as string}` specifies the parameter names and the values to pass for those parameters; a method may have no parameters
   - `SOAPAction:"urn:xmethodsBabelFish#BabelFish"` specifies the required SOAPAction value
3. It returns the result of the SOAP request. That consists of a text string that contains the translated text.

Note that AppleScript, working through the Apple Event Manager, did all the work of formatting the `call soap` script statement into proper XML, opening a connection to the specified server, sending the message, waiting for a reply, formatting the returned XML into an Apple event, and returning the event.

A More Complex Translation Script

The script in Listing 3-3 (page 24) performs the same task as the script in Listing 3-2 (page 22), translating an English phrase to French. However, it is more flexible and sophisticated in several ways:

1. It turns the translation handler into a flexible SOAP request routine that can call different SOAP servers and different methods depending on the parameters passed to it.
2. It uses error handling in the SOAP request handler and sets a boolean parameter value so that the caller can check for success. The main script checks that parameter and displays an error message if the SOAP request was unsuccessful.
3. The main script also shows how to use a `try` block to handle errors. In this case, it checks for errors while displaying a dialog: for any returned error number except -128 (the user cancelled), it beeps.
4. It defines property values to make the script more readable and easier to modify.

Listing 2-3 A more detailed script that calls a SOAP translation server.

```
-- Use properties to store default values.
property SOAP_app : "http://services.xmethods.net:80/perl/soaplite.cgi"
property method_name : "BabelFish"
property method_namespace_URI : "urn:xmethodsBabelFish"
property SOAP_action : "urn:xmethodsBabelFish#BabelFish"
property English_to_French : "en_fr"
```

```
---- Main script ----------------------------------------
--Query user for text to translate.
set this_text to "Hello my friend!"
repeat
    try
        display dialog ¬
            "Enter the text to translate into French:" ¬
            default answer this_text
        set this_text to the text returned of the result
        if this_text is not "" then
            set this_text to this_text as string
            exit repeat
        end if
    on error number error_number
        -- Don't show error if user just cancelled.
        if the error_number is -128 then error number -128
        beep
    end try
end repeat

-- Create the parameter record.
set the method_parameters to {translationmode:English_to_French, ¬
     sourcedata:this_text}

-- Call the SOAP handler.
copy my SOAP_call(SOAP_app, method_name, ¬
    method_namespace_URI, method_parameters, SOAP_action) ¬
    to {call_indicator, call_result}

-- Check for error return from SOAP handler.
if the call_indicator is false then
    beep
    display dialog "An error occurred." & return & return ¬
        & call_result buttons {"Cancel"} default button 1
```

```
else
    --Show translated text and allow user to copy it to Clipboard.
    display dialog call_result buttons {"Clipboard", "OK"} ¬
        default button 2
    if the button returned of the result is "Clipboard" then
        set the clipboard to the call_result
    end if
end if

-- SOAP translation handler ------------------------------
on SOAP_call(SOAP_app, method_name, ¬
    method_namespace_URI, method_parameters, SOAP_action)
    try
        using terms from application "http://www.apple.com/placebo"
            tell application SOAP_app
                set this_result to call soap ¬
                    {method name:method_name ¬
                        , method namespace uri:method_namespace_URI ¬
                        , parameters:method_parameters ¬
                        , SOAPAction:SOAP_action}
            end tell
        end using terms from
        return {true, this_result}
    on error error_message
        return {false, error_message}
    end try
end SOAP_call
```

Making Remote Procedure Calls From Applications

Starting with OS X version 10.1, the Apple Event Manager provides support for using the XML-RPC and SOAP protocols to make remote procedure calls from AppleScript scripts and from applications. This chapter provides sample code that show how to make remote procedure calls from applications or other code.

This chapter assumes you are familiar with the conceptual material in "Introduction to XML-RPC and SOAP Programming Guide" (page 4). To test any of the sample code shown in this book, you must have an Internet connection.

Making an XML-RPC Call

To make an XML-RPC call from an application or other code, you create an Apple event that describes the remote procedure call, use the `AESend` function to send it, and extract any returned information from the reply Apple event. This process, along with the Apple Event Manager API you use, is described in "Remote Procedure Calls From Applications" (page 13).

This section provides sample code you can use from an application, tool, or other code to send an XML-RPC request to a server. The sample code calls an Internet server that returns the state name for the passed state index. That is, for an index value of 1 it returns Alabama, for 2 it returns Alaska, for 50 it returns Wyoming, and so on.

Setting Up a Project

You can use the sample code in this section in a number of ways. The following steps show one approach: creating a C++ tool with Project Builder:

1. Launch Project Builder.
2. Choose New Project from the File menu.
3. Choose the C++ tool template.
4. Replace the code in the automatically generated main.cpp file with the sample code in Listing 4-1 (page 28), Listing 4-2 (page 30), and Listing 4-3 (page 33), in that order.
5. Use Add Frameworks from the Project menu to add `Carbon.framework` to the project.

Once you've built the tool, you can run it in Project Builder and examine its output in the Console pane, or run it in a Terminal window.

You can also build this code directly in a Terminal window with the following compile line:

```
cc -g -o xmlrpc xmlrpc.cp -framework Carbon
```

If you want to compile with a C compiler, rather than a C++ compiler, you'll have to modify the code so that all variables are declared at the beginning of functions, rather than where they are used.

Includes, Constants, and Declarations

Listing 4-1 (page 28) shows include statements, constants, and declarations to place at the beginning of your code file. This code

- includes `Carbon.h` to gain access to the scripting API it needs
- defines a simple error macro that uses `fprintf` to show an error number and line number, then exits the application
- defines constants that specify the server and method name to call to get state name information
- declares a debug function that is defined later

Listing 3-1 Includes, constants, and declarations for making an XML-RPC call.

```
#include <Carbon/Carbon.h>

// Define a simple error macro that just shows the error and line number
#define checkErr(err) \
    while (err != noErr) \
        { fprintf(stderr, "Failed at line %d, error %d\n", __LINE__, err);\
         exit(-1); }

// Define constants for the XML-RPC server and method.
static const char* serverURL = "http://betty.userland.com/RPC2";
static const char* methodName = "examples.getStateName";

// Declare our debug function.
static void dumpDebug(const char* msg, const AppleEvent* replyEvent,
    OSType debugDataKey);
```

A Main Function That Makes an XML-RPC Call

Listing 4-2 (page 30) shows a `main` function that prepares an XML-RPC Apple event, sends it, and displays information from the reply Apple event. To do so, it performs the following steps:

1. It calls `AECreateDesc` to create a target address descriptor of type `typeApplicationURL` that specifies the target for the request (a remote server). It uses the constant `serverURL` defined previously to specify the desired Internet server.
2. It calls `AECreateAppleEvent` to create an Apple event with event class `kAERPCClass`, event ID `KAEXMLRPCScheme`, and with the target address descriptor from step 1.
3. It calls `AECreateList` to create the direct object for the Apple event.
4. It calls `AEPutParamPtr` to insert the method name, using the key `keyRPCMethodName` and the constant `methodName` defined previously.
5. It calls `AECreateList` to create a list descriptor for the remote procedure call parameters.
6. It calls `AEPutPtr` to insert the only parameter for the `examples.getStateName` function, the state index, into the parameter list. It passes `typeSInt32` for the type code and an index value of 41. The index number was chosen randomly from the legal values of 1 to 50 (the fifty states).
7. It calls `AEPutParamDesc` to add the parameter list to the Apple event's direct object.
8. It calls `AEPutParamDesc` to insert the direct object into the Apple event.
9. It calls `AEPutAttributePtr` to turn on debugging by adding an attribute to the event with key `keyXMLDebuggingAttr` and the value `kAEDebugXMLDebugAll`. This step is optional.
10. It initializes a reply Apple event.
11. It calls `AESend` to send the Apple event, passing `kAEWaitReply` to indicate it will wait for a reply.
12. If the previous call succeeds, it calls `AEGetParamPtr` with key `keyDirectObject` and desired type of `typeChar` to extract the direct object of the reply Apple event. For a successful call with state index 41, the returned value is the string "South Dakota".

    It calls `fprintf` to display the state name. For example

    ```
    State is South Dakota!
    ```

    Note: Starting in OS X version 10.3, `typeChar` is deprecated in favor of one of the Unicode string types. For details, see the descriptions for `typeChar` and for the Unicode string types in *Apple Event Manager Reference*.

13. The main function then calls the function `dumpDebug`, shown in Listing 4-3 (page 33), once each to display the header and data for the original XML-RPC request and for the reply from the server. Listing 4-4 (page 34) shows the possible result of these calls.

Listing 3-2 main function to send an XML-RPC Apple event.

```
//------------------------- main ----------------------------------
// Builds and sends an XML-RPC Apple event.
// Sends the Apple event to a server that returns the state
//   name corresponding to the passed number; e.g., the number
//   5 corresponds to the state California.
// Prints the returned state name, then calls function
//   to dump debug information from reply
int main()
{
    OSErr err;
    AEDesc targetAddress;

    // Create the target address.
    //   Using type typeApplicationURL makes it a remote procedure call event.
    err = AECreateDesc(typeApplicationURL, serverURL, strlen(serverURL),
&targetAddress);
    checkErr(err);

    // Create an XML-RPC Apple event
    AppleEvent xmlrpcEvent;
    err = AECreateAppleEvent(kAERPCClass, kAEXMLRPCScheme, &targetAddress,
            kAutoGenerateReturnID, kAnyTransactionID, &xmlrpcEvent);
    checkErr(err);
    AEDisposeDesc(&targetAddress);

    // Create the parameters for the event - the direct object is a record
    //   that contains the method name, and a list of parameters
    AEDesc directObject;
    err = AECreateList(NULL, 0, true, &directObject);
    checkErr(err);
```

```
// Insert the method name
err = AEPutParamPtr(&directObject, keyRPCMethodName, typeChar,
    methodName, strlen(methodName));
checkErr(err);

// Create the list for the actual parameters
AEDesc paramList;
err = AECreateList(NULL, 0, false, &paramList);
checkErr(err);

// Put the state index into the parameter array
SInt32 stateIndex = 41; // Should correspond to South Dakota
err = AEPutPtr(&paramList, 0, typeSInt32, &stateIndex,
    sizeof(stateIndex));
checkErr(err);

// Put the parameter list into the direct object
err = AEPutParamDesc(&directObject, keyRPCMethodParam, &paramList);
checkErr(err);
AEDisposeDesc(&paramList);

// Put the direct object into the event
err = AEPutParamDesc(&xmlrpcEvent, keyDirectObject, &directObject);
checkErr(err);
AEDisposeDesc(&directObject);

// Request all available debugging information. That will include
//   the header and body for both the XML-RPC request and the reply
//   from the server.
SInt32 debugAttr = kAEDebugXMLDebugAll;
err = AEPutAttributePtr(&xmlrpcEvent, keyXMLDebuggingAttr, typeSInt32,
    &debugAttr, sizeof(debugAttr));

// Send the event
```

```
    AppleEvent replyEvent;
    AEInitializeDescInline(&replyEvent);
    err = AESend(&xmlrpcEvent, &replyEvent, kAEWaitReply, kAENormalPriority,
        kAEDefaultTimeout, NULL, NULL);
    checkErr(err);

    // The direct object of the reply Apple event contains
    // our result (the name of the state)
    char buffer[255];
    Size actualSize;
    err = AEGetParamPtr(&replyEvent, keyDirectObject, typeChar, NULL,
        buffer, sizeof(buffer), &actualSize);
    checkErr(err);

    fprintf(stderr, "State is %.*s!\n", actualSize, buffer);

    // Dump debug information
    dumpDebug("HTTP POST header", &replyEvent, keyAEPOSTHeaderData);
    dumpDebug("XML Request", &replyEvent, keyAEXMLRequestData);
    dumpDebug("HTTP Reply header", &replyEvent, keyAEReplyHeaderData);
    dumpDebug("XML Reply", &replyEvent, keyAEXMLReplyData);

    return 0;
}
```

Displaying XML-RPC Debug Information

Listing 4-3 (page 33) shows a function that extracts debug information from a passed Apple event and displays it with `fprintf`. The information displayed depends on the key passed in the `debugDataKey` parameter. To display debug data, this function:

1. Calls `AEGetParamDesc`, passing the key specified by the `debugDataKey` parameter and the desired type `typeChar`, to extract character data for that debug key.

   The possible debug keys are shown in "Apple Event Manager API for Remote Procedure Calls" (page 14).

2. If step 1 fails, it uses `fprintf` to display an error message.

Otherwise, it formats the retrieved character data to show tabs, carriage returns, and line feeds, and displays it with `fprintf`.

Listing 3-3 A function to display debug information from an XML-RPC reply Apple event.

```
// ------------------------- dumpDebug ----------------------------------
// Extract and display debug information from a reply Apple event.

static void dumpDebug(const char* msg, const AppleEvent* replyEvent,
    OSType debugDataKey)
{
    fprintf(stderr, "%s:\n", msg);

    AEDesc paramDesc;
    OSErr err = AEGetParamDesc(replyEvent, debugDataKey,
                                typeChar, &paramDesc);
    if (err != noErr)
        fprintf(stderr, "\tCan't get debug data %4.4s - %d returned\n",
            &debugDataKey, err);
    else
    {
        int len = AEGetDescDataSize(&paramDesc);
        char* buffer = new char[len];
        AEGetDescData(&paramDesc, buffer, len);

        char* p = buffer;
        char* pEnd = buffer + len;

        while (p < pEnd)
        {
            char* pNext = strpbrk(p, "\r\n");
            if (pNext == NULL)
                pNext = pEnd;
            else
            {
                while (pNext < pEnd && (*pNext == '\r' || *pNext == '\n'))
                {
                    *pNext++ = '\0';
                }
            }
            fprintf(stderr, "\t%.*s\n", pNext - p, p);
            p = pNext;
        }

        AEDisposeDesc(&paramDesc);
        delete[] buffer;
    }
    fprintf(stderr, "\n\n");
}
```

The following listing shows sample output from the `dumpDebug` function for an XML-RPC Apple event.

Listing 3-4 Sample header and data from an XML-RPC post and the reply.

```
HTTP POST header:
    POST /RPC2 HTTP/1.0
    User-Agent: OS X; AEServer (1.0)
    Host: betty.userland.com
    Content-Type: text/xml
    Content-length: 216



XML Request:
    <?xml version="1.0" encoding="UTF-8"?>
        <methodCall>
            <methodName>examples.getStateName</methodName>
            <params>
                <param>
                    <value>
                        <i4>41</i4>
                    </value>
                </param>
            </params>
        </methodCall>


HTTP Reply header:
    HTTP/1.1 200 OK
    Connection: close
    Content-Length: 141
    Content-Type: text/xml
    Date: Mon, 13 Aug 2001 03:56:34 GMT
    Server: UserLand Frontier/7.0.1-WinNT


XML Reply:
    <?xml version="1.0"?>
    <methodResponse>
        <params>
            <param>
                <value>South Dakota</value>
                </param>
            </params>
        </methodResponse>
```

Making a SOAP Request

To make a SOAP request from an application or other code, you create an Apple event that describes the request, use the `AESend` function to send it, and extract any returned information from the reply Apple event. This process, along with the Apple Event Manager API you use, is described in "Remote Procedure Calls From Applications" (page 13).

This section provides sample code for sending a SOAP request to a server. Like the sample code in "Making an XML-RPC Call" (page 27), it makes a request to an Internet server that returns the state name for the passed state index. That is, for an index value of 1 it returns Alabama, for 2 it returns Alaska, and so on. In this case, however, the sample code calls the `getStateName` method, a SOAP method on an entirely different server.

Setting Up a Project

You can use the sample code in this task in a number of ways. The following steps show how to create a C++ tool with Project Builder:

1. Launch Project Builder.
2. Choose New Project from the File menu.
3. Choose the C++ tool template.
4. Replace the code in the automatically generated main.cpp file with the sample code in Listing 4-5 (page 36), Listing 4-6 (page 38), Listing 4-7 (page 41), and Listing 4-3 (page 33), in that order.
5. Use Add Frameworks from the Project menu to add `Carbon.framework` to the project.

Once you've built the tool, you can run it in Project Builder and examine its output in the Console pane, or run it in a Terminal window.

You can also build this code directly in a Terminal window with the following compile line:

```
cc -g -o soap soap.cp -framework Carbon
```

If you want to compile with a C compiler, rather than a C++ compiler, you'll have to modify the code so that all variables are declared at the beginning of functions, rather than where they are used.

Includes, Constants, and Declarations

Listing 4-5 (page 36) shows include statements, constants, and declarations to place at the beginning of your code file. This code

- includes Carbon.h to gain access to the scripting API it needs
- defines a simple error macro that uses `fprintf` to show an error number and line number, then exits the application
- defines constants that specify the server and method name to call to get state name information
- declares a function (defined later) that creates a parameter list record
- declares a debug function that is defined later

Listing 3-5 Includes, constants, and declarations for making a SOAP request.

```
#include <Carbon/Carbon.h>
```

```
#define checkErr(err) \
    while (err != noErr) \
    { fprintf(stderr, "Failed at line %d, error %d\n", __LINE__, err); \
    exit(-1); }

static const char* serverURL = "http://www.soapware.org/examples";
static const char* methodName = "getStateName";
static const char* methodNameSpaceURI = "http://www.soapware.org/";

static OSStatus createUserRecord(AEDesc* desc, const char* paramName,
    OSType dataType, const void* data, UInt32 dataSize);
static void dumpDebug(const char* msg, const AppleEvent* reply,
    OSType debugDataKey);
```

A Main Function That Makes a SOAP Request

Listing 4-2 (page 30) shows a `main` function that prepares a SOAP request Apple event, sends it, and displays information from the reply Apple event. To do so, it performs the following steps:

1. It calls `AECreateDesc` to create a target address descriptor of type `typeApplicationURL` that specifies the target for the request (a remote server). It uses the constant `serverURL` defined previously to specify the desired Internet server.
2. It calls `AECreateAppleEvent` to create an Apple event with event class `kAERPCClass`, event ID `kAESOAPScheme`, and with the target address descriptor from step 1.
3. It calls `AECreateList` to create the direct object for the Apple event.
4. It calls `AEPutParamPtr` to insert the method name, using the key `keyRPCMethodName` and the constant `methodName` defined previously.
5. It calls the function `createUserRecord`, shown in Listing 4-7 (page 41), to create an `AEDesc` record that stores the parameter name and value for a method that has one of each.
6. It calls `AEPutParamDesc` to add the parameter record to the Apple event's direct object.
7. It calls `AEPutParamPtr` to insert the method namespace URI, using the key `keySOAPMethodNameSpaceURI` and the constant `methodNameSpaceURI` defined previously.
8. It calls `AEPutParamDesc` to insert the direct object into the Apple event.
9. It calls `AEPutAttributePtr` to turn on debugging by adding an attribute to the event with key `keyXMLDebuggingAttr` and the value `kAEDebugXMLDebugAll`. This step is optional.
10. It initializes a reply Apple event.

11. It calls `AESend` to send the Apple event, passing `kAEWaitReply` to indicate it will wait for a reply.
12. If the previous call succeeds, it calls `AEGetParamPtr` with key `keyDirectObject` and desired type of `typeChar` to extract the direct object of the reply Apple event. For a successful call with state index 41, the returned value is the string "South Dakota".

    It calls `fprintf` to display the state name. For example

    ```
        State is South Dakota!
    ```

13. The main function then calls the function `dumpDebug`, shown in Listing 4-3 (page 33), once each to display the header and data for the original XML-RPC request and for the reply from the server. Listing 4-8 (page 42) shows the possible result of these calls.

Listing 3-6 main function to send a SOAP Apple event.

```
//------------------------- main -----------------------------------
// Builds and sends a SOAP request Apple event.
// Sends the Apple event to a server that returns the state
//   name corresponding to the passed number; e.g., the number
//   5 corresponds to the state California.
// Prints the returned state name, then calls function
//   to dump debug information from reply
int main()
{
    OSErr err;
    AEDesc targetAddress;

    // Create the target address
    //   Using type typeApplicationURL makes it a remote procedure
    //   call event.
    err = AECreateDesc(typeApplicationURL, serverURL, strlen(serverURL),
        &targetAddress);
    checkErr(err);

    // Create a SOAP request Apple event
    AppleEvent event;
    err = AECreateAppleEvent(kAERPCClass, kAESOAPScheme, &targetAddress,
```

```
        kAutoGenerateReturnID, kAnyTransactionID, &event);
    checkErr(err);
    AEDisposeDesc(&targetAddress);

    // Create the parameters for the event - the direct object is a
    // record that contains the method name, and a list of parameters

    AEDesc directObject;
    err = AECreateList(NULL, 0, true, &directObject);
    checkErr(err);

    // Put the method name
    err = AEPutParamPtr(&directObject, keyRPCMethodName, typeChar,
        methodName, strlen(methodName));
    checkErr(err);

    // The parameters for a SOAP request are a record. We use the
    // "AppleScript" format for records that contain user readable
    // names, and call a helper routine to construct this record.
    AEDesc paramRecord;
    SInt32 stateIndex = 41;
    err = createUserRecord(&paramRecord, "statenum", typeSInt32,
        &stateIndex, sizeof(stateIndex));
    checkErr(err);

    // Put the parameter record into the direct object
    err = AEPutParamDesc(&directObject, keyRPCMethodParam,
        &paramRecord);
    checkErr(err);
    AEDisposeDesc(&paramRecord);

    // Additional pieces for soap are the SOAP schema, method
    //      namespaceURI and SOAPAction.
    // If the SOAP schema is not explicitly specified, it defaults
    //      to kSOAP1999Schema for the 1999 schema
```

```
//      (corresponding to the 1.1 specification).
// If the SOAPAction is not explicitly specified, it will be
//    the path part of the URL (in this case, "/examples")

err = AEPutParamPtr(&directObject, keySOAPMethodNameSpaceURI,
    typeChar, methodNameSpaceURI, strlen(methodNameSpaceURI));
checkErr(err);

// Put the direct object into the event
err = AEPutParamDesc(&event, keyDirectObject, &directObject);
checkErr(err);
AEDisposeDesc(&directObject);

// Request debugging information
SInt32 debugAttr = kAEDebugXMLDebugAll;
err = AEPutAttributePtr(&event, keyXMLDebuggingAttr, typeSInt32,
    &debugAttr, sizeof(debugAttr));

// Send the event
AppleEvent reply;
AEInitializeDescInline(&reply);
err = AESend(&event, &reply, kAEWaitReply, kAENormalPriority,
    kAEDefaultTimeout, NULL, NULL);
checkErr(err);

// The direct object contains our result (the name of the state)
char buffer[255];
Size actualSize;
err = AEGetParamPtr(&reply, keyDirectObject, typeChar, NULL,
    buffer, sizeof(buffer), &actualSize);
checkErr(err);

fprintf(stderr, "State is %.*s!\n", actualSize, buffer);
```

```
    // Dump debug information
    dumpDebug("HTTP POST header", &reply, keyAEPOSTHeaderData);
    dumpDebug("XML Request", &reply, keyAEXMLRequestData);
    dumpDebug("HTTP Reply header", &reply, keyAEReplyHeaderData);
    dumpDebug("XML Reply", &reply, keyAEXMLReplyData);

    return 0;
}
```

Creating Parameter List Records

Listing 4-7 (page 41) shows a function that creates a record containing the parameter name and value for a SOAP method with one parameter.

To prepare a parameter list record, this function performs the following steps:

1. It calls `AECreateList` to create a list descriptor record for the SOAP request parameters.
2. It calls `AECreateList` again to create a list descriptor for the parameter name and value.
3. Calls `AEPutPtr` to insert the passed parameter name in the parameter name and value list, with the type `typeChar`.
4. Calls `AEPutPtr` again to insert the passed parameter data in the parameter name and value list, with the type specified the passed type.
5. It calls `AEPutParamDesc` to add the parameter name and value list to the parameter list record.

If you need to create a parameter list record for a SOAP request with multiple parameters, you can define a function with a variable length argument list. The function can iterate over the arguments, performing steps 3 and 4 (adding the name and data) for each pair of parameter arguments.

Listing 3-7 A function to create a parameter list record for a SOAP request Apple event.

```
// -------------------- createUserRecord -----------------------------
// Creates a record containing the parameters for a SOAP request.
// Uses the "AppleScript" format for records that contain user readable
// names.

static OSStatus createUserRecord(AEDesc* desc, const char* paramName,
    OSType dataType, const void* data, UInt32 dataSize)
```

```
{
    OSErr err = AECreateList(0, NULL, true, desc);
    if (err == noErr) {
        AEDesc termsList;
        err = AECreateList(0, NULL, false, &termsList);

        if (err == noErr)
            err = AEPutPtr(&termsList, 0, typeChar,
                paramName, strlen(paramName));
        if (err == noErr)
            err = AEPutPtr(&termsList, 0, dataType,
                data, dataSize);

        if (err == noErr)
            err = AEPutParamDesc(desc, keyASUserRecordFields,
                &termsList);

        AEDisposeDesc(&termsList);
    }
    return err;
}
```

Displaying SOAP Request Debug Information

The sample code to display debug information from a SOAP request Apple event uses the same `dumpDebug` function as the sample code for making an XML-RPC call. That code is shown in Listing 4-3 (page 33). However, because the XML-RPC and SOAP protocols are different, the requests, the Apple events that represent them, and the debug output are all different. Listing Listing 4-8 (page 42) shows sample output from the `dumpDebug` function for a SOAP request Apple event.

Listing 3-8 Sample header and data from a SOAP post and the reply.

```
HTTP POST header:
    POST /examples HTTP/1.0
    Host: www.soapware.org
    SOAPAction: "/examples"
    User-Agent: OS X; AEServer (1.0)
    Content-Type: text/xml
    Content-length: 538


XML Request:
    <?xml version="1.0" encoding="UTF-8"?>
    <SOAP-ENV:Envelope
        xmlns:xsd="http://www.w3.org/1999/XMLSchema"
        xmlns:xsi="http://www.w3.org/1999/XMLSchema-instance"
        SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
        xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
        xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
        <SOAP-ENV:Body>
            <m:getStateName xmlns:m="http://www.soapware.org/">
                    <statenum xsi:type="xsd:int">41</statenum>
            </m:getStateName>
        </SOAP-ENV:Body>
    </SOAP-ENV:Envelope>


HTTP Reply header:
    HTTP/1.1 200 OK
    Connection: close
    Content-Length: 499
    Content-Type: text/xml; charset="us-ascii"
    Date: Mon, 13 Aug 2001 05:07:46 GMT
    Server: UserLand Frontier/7.0.1-WinNT


XML Reply:
    <?xml version="1.0"?>
    <SOAP-ENV:Envelope
SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
        xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:SOAP-ENV="http://
```

```
        schemas.xmlsoap.org/soap/envelope/"
xmlns:xsd="http://www.w3.org/1999/XMLSchema"

        xmlns:xsi="http://www.w3.org/1999/XMLSchema-instance">

        <SOAP-ENV:Body>

            <getStateNameResponse>

                <Result xsi:type="xsd:string">South Dakota</Result>

                </getStateNameResponse>

            </SOAP-ENV:Body>

        </SOAP-ENV:Envelope>
```

Document Revision History

This table describes the changes to *XML-RPC and SOAP Programming Guide*.

Date	Notes
2005-03-03	Made minor changes to first two chapters. Changed title from "XML-RPC and SOAP Support." In "Making Remote Procedure Calls From Scripts" (page 18), added warning that the script samples may not work because the web services they rely on may no longer be available. Also provided the location of some additional sample scripts.
2001-09-07	First version.