Using and Providing Web Services in Tcl

Gerald W. Lester
TicketSwitch USA, LLC
Overview

• What are Web Services
• Available Packages
• Traditional Approaches
• Our Approach
• Examples
• Conclusion
What are Web Services

- W3C
  - XML Based
  - SOAP based
  - Web Services Description Language (WSDL)
- Other definitions exist
- RESTful
Available Packages

- WebServices for Tcl
- Server
- TclHttpd
- Client
- Any Tcl Application

http://members.cox.net/~gerald.lester/WebServicesForTcl.html
Available Packages

- Server
  - Tcl Web Services Toolkit (TWIST)
  - AOLserver
- Client
  - None yet, future goal
- http://code.google.com/p/twsdl/
Traditional Approaches

• Define WSDL by hand
• Generate documentation for service by hand
• Use tool to generate server abstract class definition
  • Handle your own XML
• Use tool to generate client stubs
  • Handle your own XML
Our Approach

• No knowledge of XML is required
  • Dictionaries used for data structures
  • Attempt to make as much in the Tcl spirit as possible
• Typing and constraints not enforced
Our Approach - Server Side

• Server generates WSDL and “man/help” page from definition of procedure and data structures

• Web Services are strongly typed
  
  • Uses literate programming style
Our Approach - Client Side

- Client side parses WSDL
  - Optionally generates stubs
  - Parsed WSDL can be saved to a file and reloaded
    - For when WSDL is not accessible
  - More efficient
Packages Used

- tDOM
- http
- log
- uri
- html
- dict
Current Status

- Version 1.x
  - Man/Help pages
  - Tutorial (thanks Bryan Oakley)
  - [http://www.tclscripting.com/articles/nov06/article1.html](http://www.tclscripting.com/articles/nov06/article1.html)
Example

- Echo service
  - Two methods
    - SimpleEcho
      - Input: String to echo
      - Returns: String
  - ComplexEcho
    - Inputs: String to echo
    - Returns:
      - Date/time
      - String
package require WS::Client
##
## Get Definition of the offered services
##
::WS::Client::GetAndParseWsdl http://localhost:8015/service/wsExamples/wsd1

set testString "This is a test"
set inputs [list TestString $testString]
puts stdout "Calling SimpleEcho via DoCalls!"
set results [::WS::Client::DoCall wsExamples SimpleEcho $inputs]
puts stdout "\t Received: {$results}"

puts stdout "Calling ComplexEcho via DoCalls!"
set results [::WS::Client::DoCall wsExamples ComplexEcho $inputs]
puts stdout "\t Received: {$results}"
## Generate stubs and use them for the calls

::WS::Client::CreateStubs wsExamples
puts stdout "Calling SimpleEcho via Stubs!"
set results [::wsExamples::SimpleEcho $testString]
puts stdout "\t Received: {$results}"
puts stdout "Calling ComplexEcho via Stubs!"
set results [::wsExamples::ComplexEcho $testString]
puts stdout "\t Received: {$results}"
Client Example - Asynchronous

## Define asynchronously callback routines

```tcl
proc success {service operation result} {
    global waitVar
    puts stdout "A call to $operation of $service was successful and returned $result"
    set waitVar 1
}

proc hadError {service operation errorCode errorInfo} {
    global waitVar
    puts stdout "A call to $operation of $service was failed with {$errorCode} {$errorInfo}"
    set waitVar 1
}
```
Client Example - Asynchronous

```bash
##
## Call asynchronously
##

set waitVar 0
puts stdout "Calling SimpleEcho via DoAsyncCall!"
::WS::Client::DoCall wsExamples SimpleEcho $inputs \  
    [list success wsExamples SimpleEcho] \  
    [list hadError wsExamples SimpleEcho]
vwait waitVar
puts stdout "Calling ComplexEcho via DoAsyncCall!"
::WS::Client::DoCall wsExamples ComplexEcho $inputs \  
    [list success wsExamples SimpleEcho] \  
    [list hadError wsExamples SimpleEcho]
vwait waitVar
```
Client Example - Google API

package require WS::Client
package require dict

::WS::Client::GetAndParseWsdl "http://api.google.com/GoogleSearch.wsdl"

dict set args key "<your google license key here>"
dict set args q {site:tclscripting.com font}
dict set args start 0
dict set args maxResults 10
dict set args filter true
dict set args restrict {}
dict set args safeSearch false
dict set args lr {}
dict set args ie latin1
dict set args oe latin1

set result [::WS::Client::DoCall GoogleSearchService doGoogleSearch $args]

foreach item [dict get $result return resultElements item] {
    puts [dict get $item title]
    puts [dict get $item URL]
    puts ""
}

package require WS::Server
package require WS::Utils

## Define the service

::WS::Server::Service \ 
  -service wsEchoExample \ 
    -description {Echo Example - Tcl Web Services} \ 
    -host $::Config(host):$::Config(port)
Defining Schema

## Define any special types

::WS::Utils::ServiceTypeDef Server wsEchoExample echoReply {
  echoBack {type string}
  echoTS   {type dateTime}
}

Defining Operations

## Define the operations available

::WS::Server::ServiceProc \  
   wsEchoExample \  
   {SimpleEcho {type string comment {Requested Echo}}} \  
   {TestString {type string comment {The text to echo back}}} \  
   {Echo a string back} {  
      return [list SimpleEchoResult $TestString]  
   }

::WS::Server::ServiceProc \  
   wsEchoExample \  
   {ComplexEcho {type echoReply comment {Requested Echo -- text and timestamp}}} \  
   {TestString {type string comment {The text to echo back}}} \  
   {Echo a string and a timestamp back} {  
      set timeStamp [clock format [clock seconds] -format {%Y-%m-%dT%H:%M:%SZ} -gmt yes]  
      return [list ComplexEchoResult [list echoBack $TestString echoTS $timeStamp]]}
Demo of Auto-generated Pages

- Service Documentation
- WSDL
Known Problems

- Some “legal” XSchema very hard to parse
- .NET generated with some tools
- Some Server implements expect certain “hardcoded” namespace prefixes instead.
Workarounds

- For most “difficult” XSchema minor edits allow it to be parsed
- No good workaround for “hardcoded” namespace problems
Conclusion

• Provides an easy way to provide Web Services from Tcl.

• Useable to call a good number of Web Services

  • May require minor modifications to WSDL

• Consider a work in progress
Questions?