



We're the 2 in B2BSM

TierBroker Easy SOAP Now

Using TierBroker for Web Services

Universal Data Interface Corporation Technical Document

VERSION 2.0a, April 24, 2002

Copyright Notice and Non-Disclosure

© Copyright Universal Data Interface Corporation ("UDICo") 2002. All rights reserved. This material is proprietary and confidential, and is furnished under a non-disclosure agreement. UDICo requires prior written consent before this material is disclosed to any third party. The names of UDICo clients and staff are confidential and may not be disclosed without prior written consent. Material that is quoted from sources other than UDICo is the property of the copyright holder. Company and product names are trademarks or registered trademarks of their respective companies.

Table of Contents

1. TIERBROKER 5-4-1	1
1.1. 5 Utilizations for 1 Development.....	1
1.2. Self-Contained Solution.....	1
1.3. Business Friendly Pricing	1
2. LET'S GO TIERBROKER!.....	2
2.1. Getting Started	2
2.2. What are Web Services and What Are They Good For?.....	2
2.3. Web Services Make Building Systems Easier	2
2.4. Why Read this White Paper?	3
2.5. The World's Smallest SOAP Server	3
2.6. What is TierBroker?	3
2.7. Web Services are Peer to Peer not Client/Server.....	4
2.8. The World's Easiest Tutorial	4
2.9. Not Just a Pretty Face	5
2.10. Edge Computing – You're Living in Your Own Private Akamai	5
2.11. HTTP Keep-Alive – Up to 50% Faster SOAP.....	6
2.12. Relationship Between TierBroker and Web/App Server.....	6
2.13. How to Read the Instructions in This Manual	8
2.14. Reading TierBroker Files	8
2.15. Questions and Customer Support.....	8
3. CREATE TBSCRIPT SOURCE CODE	9
3.1. Create a Web Service.....	9

3.2.	What is TBScript?.....	9
3.3.	Advantages of Scripting	9
3.4.	The Business Day Calendar Class	10
3.5.	Running the BusinessDayCalendar Example	10
3.6.	Configuring Advanced SOAP Parameters	10
3.7.	The _operation Keyword Detail.....	11
3.8.	The SOAP Service Definition	11
3.9.	Saving BusinessDayCount as a Project	12
3.10.	The _volatile Keyword Detail	12
3.11.	Default Parameters in SOAP Functions	12
4.	TESTING WEB SERVICES WITH A TEST HARNESS.....	13
4.1.	Testing Web Services	13
4.2.	Testing HTML Applications	13
4.3.	Testing Application Source Code.....	14
4.4.	Run the Test Application	14
4.5.	The TierBroker Console	15
5.	TEST THE WEB SERVICE	16
5.1.	Host Your Own Web Service	16
5.2.	Easy Testing with TBSOap	16
5.3.	TierBroker Server Logging.....	17
6.	USING WITH MS EXCEL.....	18
6.1.	Version Compatibility with MS Windows and MS Office	18
6.2.	Run the TierBroker Server.....	18
6.3.	Open Source Excel Add-in	18
6.4.	Configuring MS Excel	18

6.5. Using the Excel Web Services Add-in.....	19
6.6. Attaching a SOAP Service	20
6.7. Using TierBroker as a Proxy to Other Web Services	22
7. HIGH VOLUME BATCH PROCESSING	24
7.1. High Performance Web Services.....	24
7.2. Files Used in this Example.....	24
7.3. Myth Buster – XML is Not High Performance	24
7.4. Myth Buster – Web Services are Too Slow for Batch Processing	24
7.5. Overview of the High Batch Volume Application.....	25
7.6. Create the 67MB 500,000 Record Transaction File.....	25
7.7. Create a Daily P&L Report.....	27
7.8. Daily P&L Results	28
8. EDIT WITH XML SPY	29
8.1. Using TierBroker with Third Party Products	29
8.2. XML and WSDL Editing.....	29
8.3. Create a New Soap Request	29
8.4. Edit the Request Parameters.....	30
8.5. Send the Request to the TierBroker Server.....	31
8.6. Editing the WSDL with XML Spy.....	32
9. VIEW THE WSDL WITH INTERNET EXPLORER.....	34
9.1. Using IE to Access WSDL.....	34
10. VALIDATE THE WSDL.....	35
10.1. Online WSDL Validation	35
10.2. Successful Validation	36
11. TRACE THE HTTP	37

11.1.	Using the MS Soap Toolkit	37
11.2.	Open a Formatted Trace.....	37
11.3.	Run the TB Server.....	38
11.4.	Run a Second TB Server as a Client.....	38
11.5.	View the Trace Results.....	39
12.	ADDING AN HTML FRONT END	40
12.1.	Create an HTML Page for BusinessDayCount().....	40
12.2.	Files Used in this Example	40
12.3.	Run the Server	40
12.4.	Use the HTML Form.....	41
12.5.	View the SOAP Result.....	41
13.	HTML PROXY FOR SOAP SERVICES.....	42
13.1.	Calling External Web Services from HTML	42
13.2.	Primary Files Used in this Example	42
13.3.	Other Files Used in the StockQuote Application.....	42
13.4.	To Run the StockQuote Application	42
13.5.	Understanding the StockQuote Project	43
13.6.	The StockQuote Web Service	44
13.7.	HTML Page with Proxy Call.....	44
13.8.	Creating the TierBroker Project File	45
13.9.	Rendering the SOAP Response into HTML.....	45
14.	HTML RENDERING WITH TEMPLATES.....	46
14.1.	Rendering the StockQuote Response	46
14.2.	Rendering Methods	46
14.3.	TierBroker Templates.....	46

14.4.	Indicating Rendering Steps in HTML Forms.....	47
14.5.	The StockQuote Response Template	47
14.6.	The StockQuote HTML Result.....	49
14.7.	Advanced Formatting with the TB Element	49
14.8.	Debugging HTML Applications.....	49
14.9.	Using the TEXT Element.....	49
14.10.	Including Sample Text	50
14.11.	Formatting Numbers, Dates and Lists	50
14.12.	Formatting Numbers	50
14.13.	Formatting Dates.....	52
14.14.	Indicating the Source Date Format	52
14.15.	Formatting Lists	53
14.16.	Setting HTML Attribute Values with the TB Element.....	53
15.	XSLT AND EXTERNAL PROCESSORS.....	54
15.1.	Using XSLT to Render HTML	54
15.2.	HTML Input Form with XSLT Directive	54
15.3.	The XSLT Style Sheet.....	55
15.4.	To Run the XSLT Example	56
15.5.	The HTML Result	56
16.	USING SCRIPTED FUNCTIONS TO RENDER HTML	57
16.1.	Using TBScript to Render HTML.....	57
16.2.	To Run the Scripting Example.....	57
16.3.	HTML Input Form with Scripting Directive.....	57
16.4.	The stockQuoteResponse() Function	58
16.5.	The HTML Result	58

17. COMPLEX HTML FORMS	59
17.1. AirFare Price Example	59
17.2. Files Used in this Example	59
17.3. To Run the AirFare Example	59
17.4. The HTML Input Form	60
17.5. The AirFare HTML Response	62
17.6. The getFlight() XML SOAP Response.....	63
17.7. The TierBroker AirFare Template.....	64
17.8. Flight Listings Table.....	65
17.9. Changing Date Formats.....	65
18. HTML EXTRACTION AND SCREEN SCRAPING	66
18.1. Turning Web Sites into Web Services	66
18.2. The Yahoo Travel Page.....	67
18.3. The HTTP GET Statement	67
18.4. The HTML Travel Query Response.....	69
18.5. The XML SOAP Response.....	70
18.6. Wrapping Legacy Systems as Web Services	70
18.7. The getFlight() SOAP Function.....	71
19. RENDERING HTML TABLES	74
19.1. RTStockQuote Price Example	74
19.2. Files Used in this Example	74
19.3. To Run the RTStockQuote Example	74
19.4. The HTML Input Form	74
19.5. The RTStockQuote HTML Response	76
19.6. The TierBroker RTStockQuote Template.....	77

20.	CURRENCY TRADER APPLICATION	78
20.1.	Bringing it All Together	78
20.2.	Accessing the Currency Trader Application from the Internet.....	78
20.3.	Primary Files Used in this Example	78
20.4.	Support Files Used in this Example.....	78
20.5.	Running the CcyTrader Application.....	79
20.6.	Accessing Currency Trader from the Browser	80
20.7.	Secure Login Using a One-Way Key	80
20.8.	Currency Trader Main Screen	81
20.9.	Execute a Trade.....	82
20.10.	Exception Handling at the Application Level	83
20.11.	Source Code Review for Currency Trader.....	83
20.12.	The CcyLogin.html Screen	84
20.13.	The CustomerLogin() SOAP Function	85
20.14.	Populating the XML Document.....	86
20.15.	The Complete CustomerLogin() SOAP Response	87
20.16.	The Customer Class Detail	88
20.17.	Exception Handling in TBScript.....	89
20.18.	Creating the Session Handle	90
20.19.	Management of Stateless Sessions	91
20.20.	The buyCurrency() SOAP Function	92
20.21.	The CcyTrader.udi Project File	93
20.22.	TierBroker Project Structure.....	94
20.23.	SQL Login Definitions.....	94
20.24.	Queue Definitions	95
20.25.	Lookups to External SQL Databases and Files	95

20.26.	Project Workflows.....	96
20.27.	Currency Trader Conclusions.....	96
21.	APPENDIX I: THE TIERBROKER IDE	97
21.1.	Product Overview.....	97
21.2.	Stepping Through Your Script.....	97
21.3.	Opening the Sample Project	97
21.4.	Understanding the TierBroker Project	98
21.5.	Project Summary	99
21.6.	Running the Workflow	100
21.7.	Debugging the Workflow	101
22.	APPENDIX II: TIERBROKER SERVER ARCHITECTURE	104
22.1.	TierBroker Server Overview	104
22.2.	Development Architecture	104
22.3.	Scripting and Multiple Language Support	105
23.	APPENDIX III: THE TIERBROKER PRICING MODEL	106
23.1.	Open Pricing for Open Systems	106
23.2.	The TierBroker Product Family	106
23.3.	The UDICo OEM Program	106
23.4.	UDICo Bonus – The Customer’s Bill of Rights	107
23.5.	What Our Competitors Don’t Want You to Know	107
23.6.	How to Negotiate for Enterprise Software	107
24.	APPENDIX IV: INSTALLATION FAQ AND QUICK START.....	109
24.1.	What is TierBroker?.....	109
24.2.	How Do I Get the TierBroker Software?.....	109
24.3.	What Do I Need to Install?	109

- 24.4. **Where Should I Install TierBroker?** 110
- 24.5. **What are the Hard Disk Requirements?**..... 110
- 24.6. **What are the Memory Requirements?** 110
- 24.7. **What are the Runtime Dependencies?**..... 110

- 25. **APPENDIX V: INSTALLATION OF TIERBROKER FOR WINDOWS**..... 111
 - 25.1. **Summary of Installation the Procedure** 111
 - 25.2. **TierBroker Components** 111
 - 25.3. **Installing Updates** 111
 - 25.4. **Download Installation Software** 112
 - 25.5. **Obtaining License Keys for the TierBroker Server and IDE**..... 112
 - 25.6. **TierBroker Server and IDE Licensing** 112
 - 25.7. **Installation of the TierBroker IDE with Setup.exe** 113
 - 25.8. **Installation of TierBroker Server with TB-VNNN.exe**..... 113
 - 25.9. **Modification of the Executable PATH**..... 113
 - 25.10. **Installation of the Excel Add-in** 114
 - 25.11. **Installation of the Kdb Database Engine** 114

- 26. **UDICO CONTACT INFORMATION**..... 115
 - 26.1. **Company Overview**..... 115
 - 26.2. **Contact Information**..... 115

1. TierBroker 5-4-1

1.1. 5 Utilizations for 1 Development

The TierBroker message is very easy to understand – with one interface you get a complete solution for the five most important data integration requirements.

Enterprise Application Integration (EAI) – Connections to MQ Series, TIBCO, JMS and other message oriented middleware.

High Speed Batch Processing – The ability to process flat-file formats that are character and position delimited, with mixed record structures at speeds of +10,000 records per second.

Application Connectors – Connecting real-time information and transactions to desktop applications. Just paste the Web Services URL into MS Excel to web-enable a spreadsheet. Create customer gateways for SAP, PeopleSoft and Siebel using the same plug-and-play approach.

XML, SOAP, WSDL – Access Web Services as a client. Publish your own Web Services API as a host. TierBroker makes Web Services as easy to use as the Browser

HTML and the Browser – Web Forms without programming! No CGI, Perl, JSP, ASP, ADO, .NET or J2EE. TierBroker allows you to create complex Web Forms with just plain-text HTML.

TierBroker means *leverage*, because **the code that powers your web site is the same code that runs the nightly batch**. There is no duplication of effort to support the *five key requirements* of data integration.

1.2. Self-Contained Solution

The TierBroker Server is a self-contained solution. All of the examples in this white paper use the TierBroker Server in combination with scripting. There is no compiling. There is no app server, web server or platform required.

The TierBroker Server works the same way the web works. The **Browser** is powered by HTML, which provides structure and JavaScript, which provides program logic. The **TierBroker Server** is powered by XML, which provides application structure and TBScript, which has the same syntax as JavaScript.

To use the TierBroker Server all you need is the server installation package and a TierBroker project file.

1.3. Business Friendly Pricing

We have visited many of our competitors web sites. Most of the time, we have no idea what they are selling or what it costs. At UDICo we are committed to an open policy of selling real software products that can be downloaded from our web site and a pricing model that is both easy to understand and customer friendly.

2. Let's Go TierBroker!

2.1. Getting Started

This document is a white paper for understanding Web Services that are implemented using XML and SOAP. It is also a tutorial for creating Web Services applications with TierBroker.

 It is not necessary to install TierBroker to follow the material in this white paper.

If you want to run the examples download the TierBroker Server from www.udico.com and follow the instructions in the *Installation FAQ and Quick Start* appendix of this white paper. The source code for this white paper is a Computer Based Training (CBT) module that is installed with the TierBroker Server:

tb/cbt/EasySoapNow

The exact location of the `tb` subdirectory will depend on whether the server is installed under Windows or UNIX. Under MS Windows TierBroker is installed in `\Program Files\UDICO\TierBroker\tb` and under UNIX the server is installed in `$HOME/tb`.

2.2. What are Web Services and What Are They Good For?

The *Web Services* model combines several technologies, including XML¹, WSDL, SOAP and UDDI, to create a simplified method for making remote procedure calls². Web Services can be used to construct distributed processing systems, perform real-time Enterprise Application Integration (EAI), as well as connect Web Pages to those services.



Web Services will have a big impact on EAI, but an even larger impact on the Web itself. Web Services will eliminate the hodge-podge of CGI, ASP and JSP required to connect HTML Browser pages to databases and transaction processing systems.

The Web Services model replaces technologies such as CORBA and COM with a set of open standards based upon XML. The Internet succeeded because the Browser was successful as a human-machine interface for remote applications. Web Services will succeed because SOAP is an ideal machine-machine interface for remote applications.

2.3. Web Services Make Building Systems Easier

In concept, developing applications with Web Services is as easy as using a Browser. Web Services work the same way a Browser works, but instead of linking a URL to an HTML Home Page, the URL of a Web Service is bound to its WSDL Page, which describes its SOAP interface.

¹ eXtensible Markup Language, Web Services Description Language, Simple Object Access Protocol and Universal Description, Discovery, and Integration respectively.

² Normally procedure calls are made within a single program running on a single computer. Remote Procedure Call methods provide a mechanism for procedures, or transactions, to be processed by remote systems.

2.4. Why Read this White Paper?

The *Easy Soap Now!* tutorial requires approximately 6 hours to complete. This tutorial will give you the training and the tools you need to create Web Services with the following capabilities:

- Available over the Internet or an intranet as SOAP.
- Hosted WSDL page for IDL discovery.
- Can be linked directly to desktop applications, such as Excel.
- Can be used with existing infrastructure, such as .NET or J2EE applications.
- Can be deployed in Browser-based applications using plain-text HTML.
- Able to process high-volume transaction batch files.

An equivalent course of study using ordinary tools would require:

- A BS or MS in equivalent Computer Science
- 5-10 reference books totaling more than 15 kilos of documentation
- 1-3 semesters of training in a university environment
- J2EE or .NET infrastructure valued at more than \$250,000

The real power of TierBroker is that it delivers the most important promise of Web Services – develop a single API and publish it as WSDL. This API can be reused across multiple and diverse applications.

2.5. The World's Smallest SOAP Server

All it takes to create a complete Web Service is 3 lines of TBScript.

```
_operation int BusinessDayCount( Date first, Date last )  
{  
    return calendar.busDayCount( first, last );  
}
```

That's all it takes to deploy a real-time SOAP application that also serves a Web Services Description Language (WSDL) page, which describes its interface to other applications. This white paper demonstrates how to use this Web Service, as well as other more complex examples.

2.6. What is TierBroker?

TierBroker is a small footprint *server* for Web Services. **TierBroker is a Web Services Server!**



TierBroker Server is a fully self-contained, platform independent executable that allows you to build, manage and deploy Web Services.

The TierBroker Server is a middleware product that lets you easily design simple or complex *workflows* between a wide choice of data resources and then lets you run those workflows at high speed, in real time, on a variety of platforms.

A **Web Server** takes plain-text HTML, which provides document structure, and JavaScript, which provides application logic, and serves those pages dynamically.

The **TierBroker Server** takes XML Project definitions, which provide workflow structure, and TBScript, which provides application logic, and hosts Web Services dynamically.

TierBroker is similar to a web server, because it runs independently of its client applications and can execute scripts, which instruct it to perform specific batch or real time functions. Unlike most middleware, the TierBroker Server can run on something as small as Windows 95 laptop or as large as a Sun E-10000 UNIX server. This makes TierBroker ideal for client side computing.

☞ Unlike a Web Server, TierBroker can run on the client as well as the server.

Web Services solves the problem of how to deliver functionality, not just Web Pages, to remote clients. TierBroker makes this easy by giving clients a Reader Application that does for Web Services, what Adobe's Acrobat Reader has done for document publishing.

2.7. **Web Services are Peer to Peer not Client/Server**

Companies that wish to adopt Web Services must find a way to deliver them to their clients. Their clients environments will be a mix of Windows and UNIX. TierBroker is an ideal delivery platform for Web Services, because it can be installed as a plug-in to applications as small as Excel and as large as PeopleSoft and SAP. On the data center side, TierBroker is a host for Web Services. On the client side, TierBroker connects those Web Services to applications that can be as simple as Excel, or complex as SAP, PeopleSoft, Siebel or BroadVision.

2.8. **The World's Easiest Tutorial**

This White Paper is also a tutorial that walks through every aspect of Web Services deployment.

Test the Source Code. *Try before you buy* – verify the application works as a batch application before deploying it as a real-time Web Service.

Test the Web Service. Use the *TBSoap* client application to verify that your Web Service can be accessed over HTTP using SOAP.

Use with MS Excel SOAP Client. Call your SOAP enabled functions from MS Excel in just 2 easy steps.

Edit with XML Spy. Call the Web Service function from XML Spy and see the results. Edit the WSDL page hosted by the TierBroker server.

View the WSDL with Internet Explorer. Use IE to view the WSDL that defines that Web Service interface.

Validate the WSDL. Validate the WSDL document with the ActiveState online application.

Trace the HTTP. Use the MS SOAP Toolkit trace utility to view the XML transferred between the client and server applications.

HTML Front End. Create an HTML front end that calls the Web Service with a plain-text HTML FORM, no programming required!

HTML Proxy. Use TierBroker as a *Web Services Request Broker* to connect HTML pages to Web Services that are not HTTP GET/POST enabled.

HTML Rendering. Use TierBroker to render the SOAP response using XSLT, HTML Templates, and Scripting.

Complex HTML Forms. Create complex forms that access SOAP requests that use nested XML schemas for input and return nested documents as output.

HTML Tables. Render complex returns into tables in just one easy step.

Currency Trader Application. A complete application with secure login procedures, stateless sessions, exception handling, transaction processing and more!

Web Services are a standards based technology. TierBroker makes it easy to learn how to create and deploy real-time applications that deliver production grade performance.

2.9. *Not Just a Pretty Face*

This white paper does not review many important features of TierBroker that are under the hood.

- The TierBroker Integrated Development Environment (IDE), which is similar to Visual Studio.
- Advanced application programming using TierBroker Projects.
- Linking TierBroker to external applications through DLL's in Windows and Shared Libraries on UNIX platforms.
- Using the TierBroker Programmer's API to call TierBroker from other applications written in languages such as C++ and Visual Basic.
- Using the TierBroker JNI and JMS interfaces to access TierBroker from Java.
- High performance flat file parsing, databases, messaging middleware and other built-in TierBroker connectivity tools.
- Operations, security, multi-threading and scalability.

TierBroker is a complete solution that has been developed and deployed at some of the largest financial institutions in the world – for additional information on TierBroker, consult the on-line documentation and Computer Based Training (CBT) modules that are shipped with the TierBroker Server.

2.10. *Edge Computing – You're Living in Your Own Private Akamai*

Edge Computing is the ability to cache information close to the client application in order to achieve a significant improvement in performance.

The impact of the TierBroker Server edge computing strategy cannot be overstated. SOAP calls over the Internet can take up to **three seconds per function**. Using TierBroker's multiplexing strategy can easily achieve **6 transactions per second**, a performance **improvement of 2000%**. Using client-side caching increases the performance up to **11,000 transactions per second**, which is a performance improvement of **nearly 5 orders of magnitude**. See *also* `tb/cbt/soap/caching.xls`.

TierBroker takes edge computing right to the client partition. In order to make SOAP function calls efficient, caching parameters may be set for each *port type*, *i.e.*, function call, that is exposed in a SOAP API. These techniques are demonstrated in the `tb/cbt/Soap` tutorial, and an *Advanced Web Services Applications Guide* is under development for Q2/2002.

☞ Some functions, such as `getCustomerNumber()`, are static, while other functions, such as `purchaseItem()` are volatile and must be executed every time. TierBroker allows you to configure caching parameters for each SOAP port type.

For additional information on SOAP parameters, see the on-line TierBroker Server Help:

UDIModel Classes/SoapEnvelope.html
UDIModel Classes/SoapRequest.html

These TierBroker Project classes specify attributes that can be used to control the SOAP port type at a fine level of granularity.

2.11. HTTP Keep-Alive – Up to 50% Faster SOAP

The TierBroker Server has fined grained control over HTTP Keep-Alive processing.

☞ The Keep-Alive tag in the HTTP header specifies if the socket connection will be left open after the current transaction. Applications that are HTTP Keep-Alive aware, can be **up to 50% faster** than SOAP applications that do not use this feature.

The project level Keep-Alive parameter settings are reviewed in the online HTML server documentation. The TierBroker Server implements the same timeout and request thresholds used by the Apache web server. In addition to control over timeout seconds and maximum requests, the application can also determine whether or not Keep-Alive should be permitted for HTTP POST transactions separately from GET requests in order to preserve Keep-Alive for HTML page processing in cases where this is disallowed for HTML POST and SOAP.

2.12. Relationship Between TierBroker and Web/App Server

TierBroker has many features typically found in Web Servers and App Servers. Browser based applications can be constructed using TierBroker alone, but the real strength of TierBroker is that it can be used tactically to enhance or extend existing infrastructure.

☞ TierBroker can be used in conjunction with ASP/.NET and/or J2EE – it does not take over the IT shop.

In cases where TierBroker is used with an existing Web Server or App Server, it will typically be interfaced to the existing front end using one of three approaches:

HTML Frames – The Browser will use TierBroker as the ACTION of selected GET and POST operations. This is the easiest way to connect a Browser based application to Web Services.

Java API – A servlet or applet will call TierBroker using the TierBroker JNI interface. The TierBroker JMS interface can also be used.

COM/DLL – A VBScript or JScript function can call to TierBroker using the TierBroker DLL, also known as the TB API.

TierBroker is also small enough to be installed on the server or the client side.

☞ **TierBroker solves the .NET/J2EE client dilemma.** TierBroker can be packaged on an OEM basis to give clients an ability to connect their applications to Web Services no matter what platform they are running.

TierBroker is a great solution for companies that want to offer Web Services to differentiate their product offerings. Server modules can be deployed at client locations to interface to Web Services with packages as small as MS Excel and as large as SAP.

2.13. *How to Read the Instructions in This Manual*

Computer source code and operating system console (DOS prompt) commands are indicated:

```
tb -UnStockQuote.udi -APlocalhost:2083 -ld3
```

The above command should be entered at the operating system console, also know as the DOS prompt.

```
TierBroker V4.21
(C) 1993-2002 Universal Data Interface Inc.
Loading Project:Version/Workflow<Easy SOAP Now:/>
Evaluation version licensed for non-commercial use.
Expires on 04/01/2002

tb>
```

Information with the TierBroker prompt 'tb>' indicates that this is displayed in the TierBroker console, after it has been launched from the DOS command line.

2.14. *Reading TierBroker Files*

TierBroker source code is a combination of XML and scripting. The TierBroker Integrated Development Environment (IDE) is an ideal tool for this tutorial, however, any text editor can be used for TierBroker development, just like any text editor can be used to edit HTML, XML or SQL.

2.15. *Questions and Customer Support*

For questions about this material contact UDICo at support@udico.com or call 212-607-7623 during US eastern standard time business hours.

3. Create TBScript Source Code

3.1. Create a Web Service

This white paper demonstrates how to build, test, and use this web service as a real-time SOAP application that also serves a Web Services Description Language (WSDL) page, describing its interface to other applications. The following source code is contained in `BusinessDayCalendar.ts`.

```
_operation int BusinessDayCount( Date first, Date last )
{
    return calendar.busDayCount( first, last );
}
```

`BusinessDayCount()` is an ordinary TBScript function. The `_operation` keyword is all that is required to publish this function as a Web Service using SOAP, and to make it available for discovery using WSDL. This function is also easy to test because it can be called from a test application before it is deployed as a real-time Web Service.

3.2. What is TBScript?

TBScript is the programming language of TierBroker.

☞ TBScript is to TierBroker as SQL is to Oracle.

The TierBroker server can execute *projects* that define *workflow*. In this example, a project is not required. Instead a Web Service is defined by a simple TBScript function.

☞ TBScript is a *curly brace* scripting language. If you know C, C++, C#, Java or JavaScript, then you already know *TBScript!*

TBScript is not a compiled language, just like SQL is not a compiled language. Instead, the TierBroker server executes scripts the same way a SQL server executes SQL. It does not matter whether the server is running on Windows or UNIX, or in batch mode or as a real-time service.

3.3. Advantages of Scripting

There are important advantages to a scripted approach to Web Services.

No Dependencies – all you need is the TierBroker server. That's it!

No Compiling – TierBroker projects and TBScript source code are not compiled, which simplifies the process of developing applications and migrating them to production.

High Level – No need to do programming for sockets, XML, SOAP, WSDL, ODBC or file parsing, because all those functions are built-in.

Less Code – A few lines of TBScript can do the work of hundreds of lines of Java or C++.

Platform Independence – TierBroker applications run transparently under Windows and UNIX and across databases such as Sybase and Oracle.

High Performance – TierBroker is typically 10 to 100 times faster than hand coded applications.

While a scripted approach does not guarantee high performance, the TierBroker Server has been used to implement some of the largest production systems in capital markets banking during its development at PricewaterhouseCoopers from 1993-2000. TierBroker applications will generally be faster than C++ or Java, because the server is optimized for very high speed parsing, database I/O, messaging and real-time transaction processing.

3.4. *The Business Day Calendar Class*

This function uses the built-in `TBCalendar` class to calculate the business day count between two dates. This class is used for business day calculations involving a single or multiple intersecting business day calendars by region code.

 The `tb/cbt/testsuite/Calendar.ts` CBT module contains an example of how to use the holiday calendar in TierBroker. The default calendar is pre-loaded with US federal holiday through 2005.

For financial calculations involving settlement periods or accruals the region code will typically be the ISO currency code. These country level regions can be subdivided by adding an extension. For example, `EUR:GBP` to indicate the British holiday calendar for a Euro based transaction sited in Great Britain. In other fields, such as energy risk management, regions will refer to geographical areas, such as *ERCOT* for Energy Resource Council of Texas.

3.5. *Running the BusinessDayCalendar Example*

The remainder of this chapter summarizes how to configure SOAP and Web Services parameters using the TBScript language.



Skip to Chapter 3 *Test the Code* to shorten this tutorial.

The following material explains how SOAP operations can be parameterized at a fine level of granularity. It is not necessary to read or understand these concepts in order to complete this tutorial.

3.6. *Configuring Advanced SOAP Parameters*

The `_operation` and `_volatile` Keywords

The keyword `_operation` indicates that this function is published as a SOAP operation for this WSDL service name, which defaults to the source code file name loaded at runtime. If the `_operation` keyword is followed by the `_volatile` keyword then response caching is disabled, and the function is always invoked. Note that an `_operation` is a normal TBScript function, and that it can be called from other functions.

Additional SOAP and WSDL Parameters

The *Easy SOAP Now* tutorial focuses on the simplest way to implement Web Services using TierBroker. However, there are more than 70 parameters which can be configured for the SOAP transaction. To review these parameters, access the online TierBroker Server help:

```
\Program Files\UDICo\TierBroker\tb\bin\cmd\english\TBServerHelp.html
```

Select *UDIModel Classes*, and then review the *SoapEnvelope*, *SoapRequest*, and *SoapParamList* classes. These parameters can be set as elements in a TierBroker Project file.

Saving a .ts File as a TierBroker Project

There are many parameters that can be used to configure a TierBroker web service. These parameters can be configured in a TierBroker project file. To save this example as a project file:

```
tb -UnBusinessDayCalendar.ts
```

This step loads the TierBroker Server with the example source file. Now save it as a TierBroker project file with the following command.

```
tb> save BusinessDayCalendar.udi
```

The *BusinessDayCalendar.udi* file can be configured with additional parameters that affect the way the Web Service operates.

3.7. The *_operation* Keyword Detail

The *_operation* keyword does not change the TierBroker function internally. It is still an ordinary TBScript function, which can be called by other functions.

 The *_operation* keyword indicates that a function should be published as a SOAP port type for *this* service.

The *_operation* keyword is also a short-hand that sets the TierBroker project value *SoapEnvelope::refreshSec* to 3600 (10 minutes) for *this* function.

3.8. The SOAP Service Definition

In the *BusinessDayCalendar.ts* example, the name of the SOAP service defaults to the source code file name. All functions that are published in this file, as well as all functions that are included from other files, will default to the SOAP service *BusinessDayCalendar*.

```
http://localhost:2083/BusinessDayCalendar/BusinessDayCount
```

This HTTP end point is constructed from the following components:

```
http://localhost:2083 // The IP address and exposed port number
BusinessDayCalendar // The SOAP Service, in this case the file name
BusinessDayCount // The name of the function being called
```

These definitions can be overridden by setting the values in a TierBroker project, and including the `BusinessDayCount` source code.

3.9. Saving *BusinessDayCount* as a Project

To save the `BusinessDayCount` example as a TierBroker project follow these steps, starting from the operating system console. Note – do not type in the comments indicated the `//` double slash.

```
cd tb/cbt/EasySoapNow           // CD to the EasySoapNow tutorial directory
tb -UnBusinessDayCalendar.ts    // Launch the TB Server from the DOS command line
```

Then, from the TierBroker console enter:

```
tb>save BusinessDayCalendar.udi // Issue the save command from the TB console
```

The TierBroker console should respond:

```
File<BusinessDayCalendar.udi> saved
```

The TierBroker project `BusinessDayCalendar.udi` has been created and can be customized to set SOAP attributes at a fine level of granularity.

3.10. The *_volatile* Keyword Detail

The `_volatile` keyword may follow the `_operation` keyword and indicates that function results should never be cached.

```
_operation _volatile int GetAccountBalance( VarChar accountCode );
```

In this example, the `GetAccountBalance()` function will always be called by TierBroker.

3.11. Default Parameters in SOAP Functions

Default parameters may be defined in the TBScript function call by assigning a value to the parameter in the function definition.

```
_operation isBusinessDay( Date date, VarChar country = 'USA:Federal' )
{
    return calendar.isBusDay( date, country );
}
```

In this case, the `country` parameter may be omitted from the SOAP request body.

4. Testing Web Services with a Test Harness

4.1. Testing Web Services

All code needs to be tested. In the case of `BusinessDayCount ()` the function is small enough that it should work the first time. However, even if it works, it may not return the correct result for several reasons that cannot be determined from an inspection of the source code alone.

- The default calendar includes US federal holidays from 2000-2005 only. Dates outside of the range will return incorrect values.
- Some US federal holidays are not observed in all financial markets.
- The application may require a holiday calendar other than US federal holidays.

This type of testing requires a range of input values for the first and last date, as well as a reconciliation to the expected results in the target application. In order to make this type of testing practical, TierBroker supports the following methods.

- Web Services functions are ordinary functions in TBScript, and can be called from standalone test programs. This technique is demonstrated in this chapter.
- The TBSOap application can be used to send batches of SOAP requests to the TierBroker server, simulating activity from a remote client. This technique is demonstrated in the following chapter.

TierBroker supports multiple testing methods that can be used as a part of the complete application deliverable. Automated testing and reconciliation is a fundamental design goal of TierBroker.

4.2. Testing HTML Applications

Readers who are primarily interested in developing HTML applications can skip the sections of the tutorial related to SOAP transaction validation.



Web Page designers can skip to Chapter 11 *Adding an HTML Front End* to shorten this tutorial.

It is not necessary to understand programming with TBScript to develop HTML front ends for TierBroker applications. A Web Page developer can use applications like *DreamWeaver* or *FrontPage* to create HTML pages that contain TierBroker directives. In this case, the Web Page developer should work with a TierBroker developer, who will create the Web Services that are the back end of the HTML.

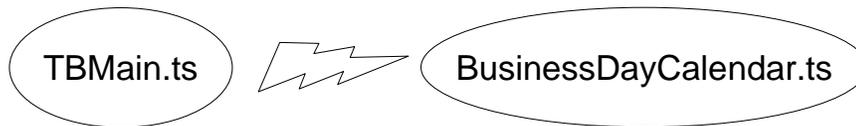
4.3. Testing Application Source Code

The following code is contained in `TBMain.ts`.

```
#include 'BusinessDayCalendar.ts'

void main()
{
    syslog << BusinessDayCount( '1/7/2002', '1/31/2002' ) << cr;
}
```

This test application calls the `BusinessDayCount ()` function to verify that it has the expected behavior.



It is easier to debug the application using the TierBroker console, than to try and diagnose a problem through a Web Services client, such as Excel.

4.4. Run the Test Application

From the operating system command line³ enter:

```
cd tb\cvt\EasySoapNow // Move to the correct subdirectory
tb -UnTBMain.ts // Run the application
```

Note – do not type in the comments, which begin with `//` and continue to the end of line.

☞ If the TierBroker server does not execute correctly, verify that the executable `PATH` contains the `C:\Progra~1\UDICo\TierBroker\tb\bin` subdirectory.

Trouble shooting information is also contained in the *Quick Start* appendix of this tutorial.

³ In this tutorial Windows is assumed to be the server operating system. The execution steps for UNIX are the same, with the exception that some operating system console commands should be modified.

4.5. The TierBroker Console

The console is positioned at the start of function `main()`:

```
TierBroker V4.21
(C) 1993-2002 Universal Data Interface Inc.
Evaluation version licensed for non-commercial use.
Expires on 03/01/2002

03. void main()
04. {
*05.     syslog << BusinessDayCount( '1/7/2002', '1/31/2002' ) << cr;
06. }

tb>
```

The asterisk `*` indicates that the server is positioned on line 5 in the source file. At the `tb>` prompt type `n` for *next* to execute the next statement.

```
19
03. void main()
04. {
05.     syslog << BusinessDayCount( '1/7/2002', '1/31/2002' ) << cr;
*06. }

tb>
```

The console shows 19 business days between January 7th and 31st. Type `go` to run the application to completion and return to the operating system prompt.

5. Test the Web Service

5.1. Host Your Own Web Service

Now that the application is tested, it can be deployed as a real-time Web Service.

```
// In the tb/cbt/EasySoapNow directory
tb -UnBusinessDayCalendar.ts -Alocalhost:2083 -ld3
```

This command line runs the server and loads the application source code. It also exposes a URL `http://localhost:2083` for SOAP and TBApi⁴ commands. The `-ld3` parameter turns on full HTTP logging so that it will be clear when the server is sending or receiving data.

Real-time Server Operation

The server console is waiting for a command. These commands can be received as SOAP through HTTP, as API commands, or directly from the console. The console can also be accessed as a Browser application, which simplifies operations when the server is deployed in a data center.

```
TierBroker V4.21
(C) 1993-2002 Universal Data Interface Inc.
Evaluation version licensed for non-commercial use.
Expires on 03/01/2002

tb>
```

The server is waiting for Web Services! In the following sections of this white paper, the `BusinessDayCount()` function is accessed using a variety of tools and methods.

5.2. Easy Testing with TBSOap

The TBSOap application uses test scripts to send commands to the TierBroker server. The output of TBSOap is an XML file that contains the test results. The `TBSOap.cmd` script file contains:

```
# Call the BusinessDayCount function
BusinessDayCount, 1/7/2002, 1/31/2002
```

To execute this Web Services test, first verify that the TierBroker server is running. Launch a second operating system shell and `cd` to the same directory. Then type the following command from the second command prompt:

```
// From the tb/cbt/EasySoapNow directory
TBSOap -iTBSOap.cmd -oTBSOap.xml
```

⁴ The TierBroker Application Programmer's Interface, or TBApi, can be used to program the server directly. This is similar to using the Oracle OCI library, or Sybase Client Library.

The TBSOap window will display:

```
G:\tb\cbt\EasySoapNow>tbssoap -iTBSOap.cmd -oTBSOap.xml
TBSOap V1.03 (C) UDICo 2001-2002

<TBSOapList>

<!-- Call the BusinessDayCount function -->
<TBSOap id = 'BusinessDayCount'
  request = '1/7/2002, 1/31/2002'
  response = '19' />

</TBSOapList>

G:\tb\cbt\EasySoapNow>
```

The XML portion of this test is captured in the output file TBSOap.xml. Saving test file results as XML files makes it easy to verify that server is responding correctly.

5.3. TierBroker Server Logging

Meanwhile, the TierBroker has the following log. This can be viewed in the console window, or by exiting the server (type exit or \ for short). The output log is saved in tb/cbt/output/output.txt.

```
<HTTPRecv><![CDATA[
POST /BusinessDayCalendar HTTP/1.1
Host: localhost
Content-Type: text/xml; charset=utf-8
Content-Length: 281
SOAPAction: "BusinessDayCalendar/BusinessDayCount "

<SOAP-ENV:Envelope xmlns:SOAP-ENV='http://schemas.xmlsoap.org/soap/envelope/'>
  <SOAP-ENV:Body>
    <BusinessDayCount xmlns="http://localhost:2083/BusinessDayCalendar">
      <first>1/7/2002</first>
      <last>1/31/2002</last>
    </BusinessDayCount>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
]]></HTTPRecv>

<HTTPSend><![CDATA[
HTTP/1.1 200 OK
Date: 01/22/2002
Content-Type: text/xml; charset=utf-8
Server: TierBroker/4.09
Connection: Keep-Alive
Content-Length: 267

<SOAP-ENV:Envelope xmlns:SOAP-ENV='http://schemas.xmlsoap.org/soap/envelope/'>
  <SOAP-ENV:Body>
    <BusinessDayCountResponse xmlns="http://localhost:2083/BusinessDayCalendar">
      <Result>19</Result>
    </BusinessDayCountResponse>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
]]></HTTPSend>
```

The output log, together with the data from the TBSOap client, verifies the correct operation of the server. The BusinessDayCount Web Service is ready to use with applications such as MS Excel, and XML Spy.

6. Using with MS Excel

6.1. Version Compatibility with MS Windows and MS Office

At UDICo we like to say that TierBroker is compatible with more versions of Windows than Microsoft. What we mean is that the following example works with MS Excel for Windows 95, 97, 98, ME, NT, 2000 and now XP. However, MS .NET applications work with Windows 2000 and XP only.

☞ TierBroker is an ideal tool to web enable computers that are using any version of Microsoft Windows, MS Office or Internet Explorer.

The fact that TierBroker can be deployed across Windows operating systems and across versions of MS Office means that Web Services can be deployed across an enterprise – not just to new computers. This is especially important since many companies are a mix of old and new computer systems.

6.2. Run the TierBroker Server

The TierBroker Server should be running. This section continues from the previous chapter.

```
// In the tb/cbt/EasySoapNow directory
tb -UnBusinessDayCalendar.ts -Alocalhost:2083 -ld3
```

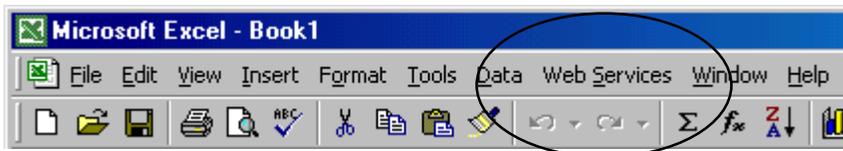
Runs the TierBroker Server and loads BusinessDayCalendar application.

6.3. Open Source Excel Add-in

The TierBroker Excel Add-in is delivered as Open Source VBA. The Excel Add-in source code can be used as a model for building more a more complex interface, or for customizing the add-in for specific applications. The Excel Add-in itself is an easy to understand example of how to use the TierBroker Programmer's API for calling the TierBroker Server from external applications.

6.4. Configuring MS Excel

The TierBroker Web Services Add-in



If MS Excel has Web Services on the main menu, then the TierBroker Web Services Add-in for MS Excel has been installed.

Installing the TierBroker Web Services Add-in

If the Excel TierBroker Web Services Add-in has not been installed, then:

1. Select Tools-Add-Ins from the main menu.

2. Select Browse and choose \Program Files\tb\cbt\Soap\UDICo.xla

These steps install the TierBroker Web Services Add-in for MS Excel.

Alternatives to the Excel Add-in

The TierBroker Excel Add-in uses the TierBroker Programmer's API, which is a DLL under Windows and a shared library under UNIX. A JNI version is also available for Java.

☞ The TierBroker Excel Add-in is provided on an open source basis, so it is easy to modify or write alternative access methods.

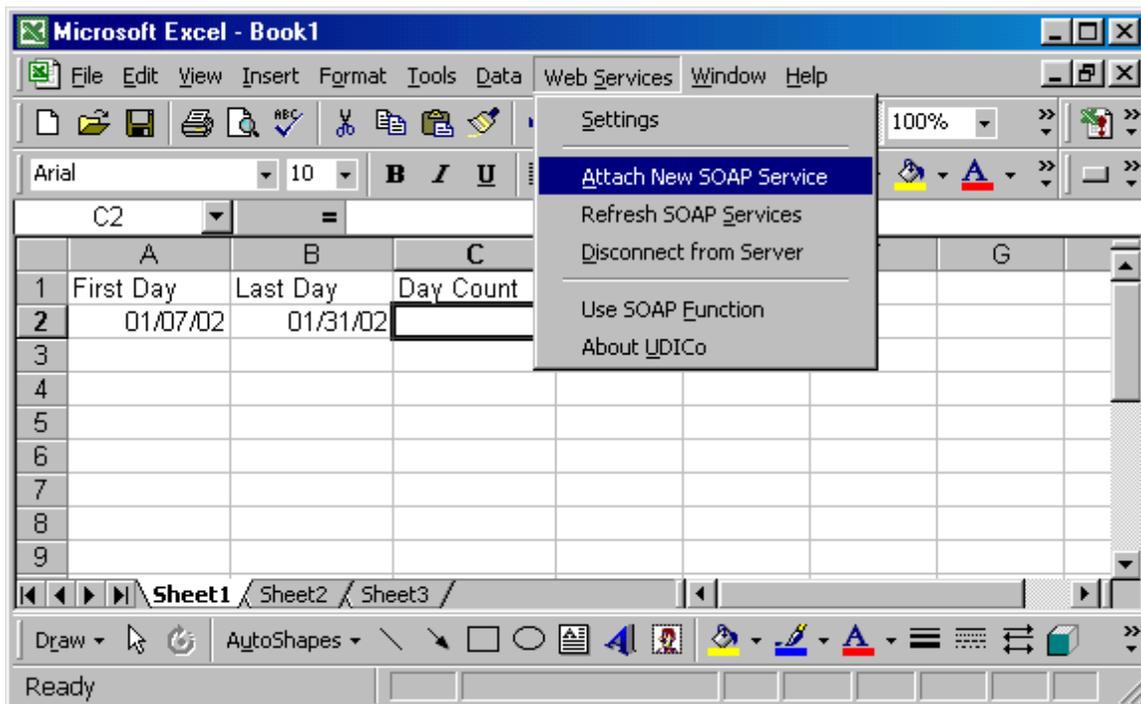
TierBroker can connect to any application, not just Excel. However, Excel is a good demonstration of how just a few lines of VBA can connect any MS Office application to the TierBroker Server.

6.5. Using the Excel Web Services Add-in

This example begins with a blank spreadsheet. Enter a *first day* and a *last day* before attaching the BusinessDayCount Web Service. The spreadsheet used in this example can also be found in:

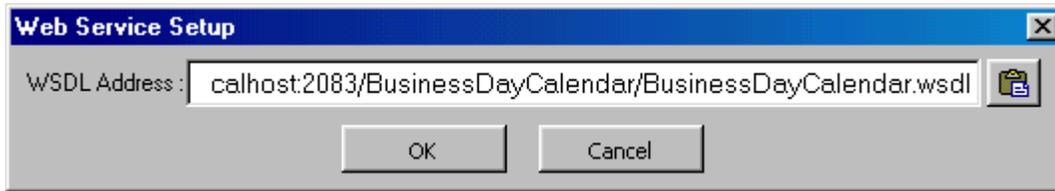
tb/cbt/EasySoapNow/BusinessDayCount.xls

The TierBroker install program creates a Web Services menu entry in Excel. To access the BusinessDayCount () function simply attach that as a new Web Service.



6.6. Attaching a SOAP Service

Select Attach New Service from the main menu.



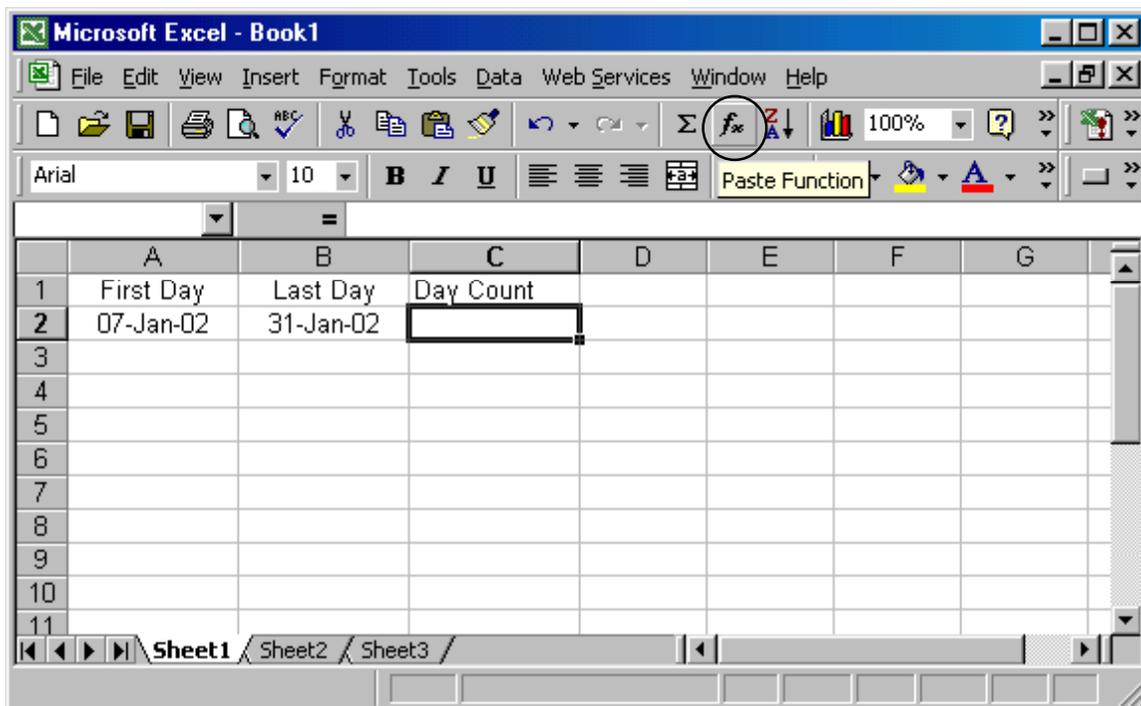
The WSDL address is:

`http://localhost:2083/BusinessDayCalendar/BusinessDayCalendar.wsdl`

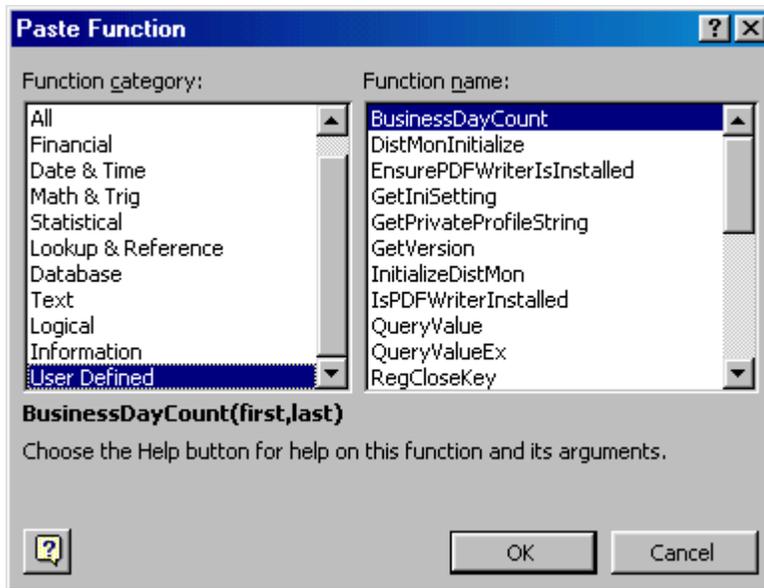
The WSDL address is the name of the service, which in this case is the name of the TBScript file, followed by the name of the service with a wsdl file extension.



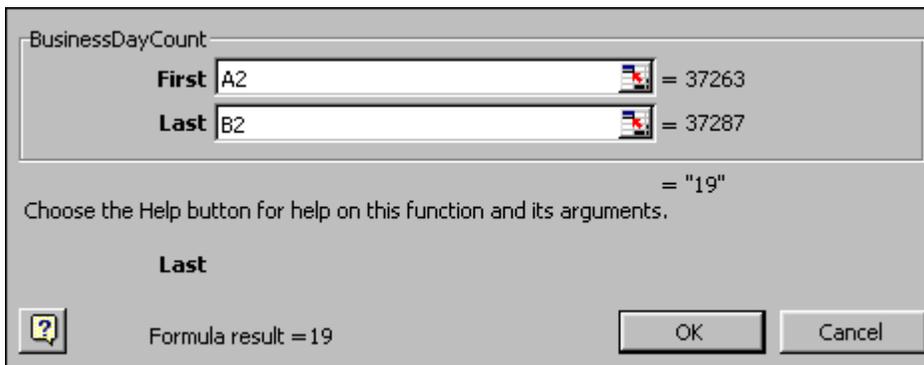
This screen indicates that the Web Service has been added and is ready to use in Excel. Simply press the **f** function icon indicated on the screen below:



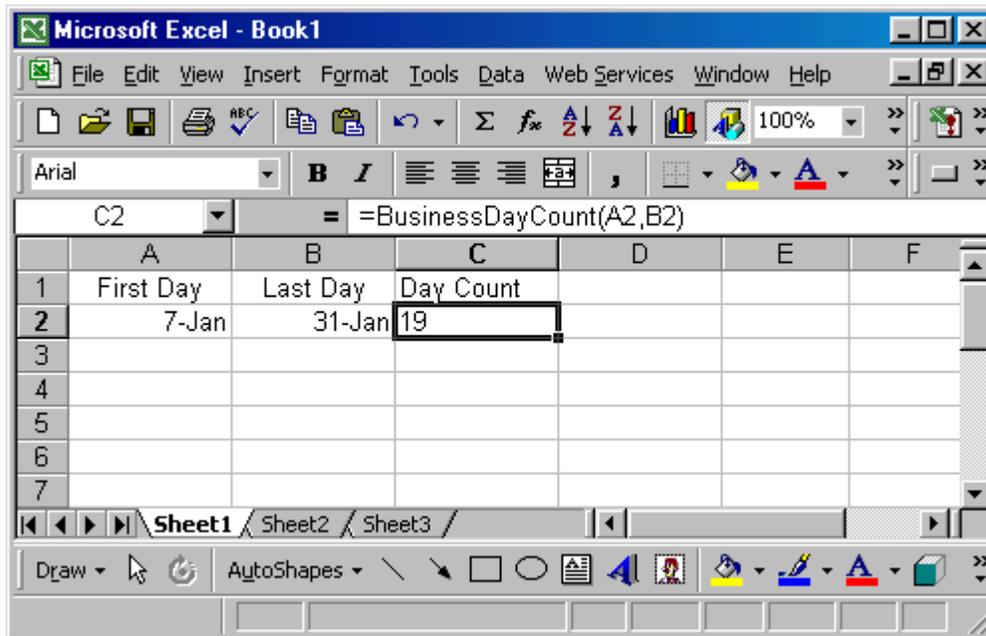
From the User Defined functions select BusinessDayCount:



Finally, select cells A2 and B2 as the sample input.



The formula result is returned indicating that the call was successful. The final screen should display the result of the business day count calculation.



That's all it takes! MS Excel can be connected to Web Services by pasting in the URI of the WSDL page, the same way a Browser accesses a web page.

6.7. Using TierBroker as a Proxy to Other Web Services

To attach other Web Services to Excel use the CBT module:

```
// Load the tbsoap spreadsheet found in
// the Soap CBT Tutorial

tb/cbt/Soap/tbsoap.xls
```

The Soap computer based training module demonstrates more advanced applications, and contains a listing of more than 30 Web Services that have been tested with TierBroker. These services are also listed on www.xmethods.org, and are hosted by a variety of tools and applications.

TierBroker has been tested with more than 60 sample Web Services to ensure that it is compatible with .NET and J2EE applications.

The AirportTemperature example hosted by CapeScience is a good example of how to link other external Web Services to TierBroker. The procedure is the same as the one used in this example.

Testing with Other SOAP Services

Many of the Web Services listed on xmethods.org are product demonstrations that have been developed by software companies to promote their products. This is the same level of grass roots support and growth that fueled the development of web sites using HTML in the early 1990's. However, it also means that most of these services are not production systems. Not every SOAP service that is listed in the tbsoap

spreadsheet will work on a consistent basis. Remember that these are primarily demonstrations of capability, so be prepared to troubleshoot new applications.

7. High Volume Batch Processing

7.1. High Performance Web Services

The TierBroker engine is a high performance Extract Transform Load (ETL) tool that has been used to implement some of the largest systems in capital markets history. This tutorial does not review how to use TierBroker for traditional data integration applications. However, a short demonstration of how TierBroker can be used to process large transactions volumes is helpful, in order to understand the role of TierBroker in the corporate data center.



It is not necessary to perform the High Volume Batch Processing example to understand how TierBroker is used to build Web Services applications. Skip this chapter to shorten the tutorial.

This chapter is valuable for anyone who is interested in using Web Services in back office applications. This module is taken from the `tb/cbt/Soap` tutorial, and the `tb/cbt/Soap/tbsoap.xls` spreadsheet can be used to see how TierBroker can revalue a portfolio of stocks in Excel by going to the Internet to pull back real time price quotes using SOAP functions.

7.2. Files Used in this Example

```
soap.h           // Class Declarations
soap.udl         // TierBroker Project File
soapreqlist.udl // HTTP End Points for SOAP Services
stocks.csv      // Sample Data
```

This tutorial example does not go into details regarding this project structure. Instead, an overview of the application is given to highlight the performance characteristics of TierBroker.

7.3. Myth Buster – XML is Not High Performance

In this sample application TierBroker will be used to generate a file of 500,000 Equity trades, using the spreadsheet data in the preceding chapter. This file is more than 67MB of XML and is created in less than 45 seconds on most Windows computers.

7.4. Myth Buster – Web Services are Too Slow for Batch Processing

In the second example, the 500,000 record batch is parsed, inserted into a database, and then repriced using real time stock quotes that are accessed using SOAP over the Internet. This step takes less than 90 seconds on most Windows computers.

7.5. Overview of the High Batch Volume Application

The EquityTrade class object:

```
// From soap.h
Class EquityTrade {
    VarChar(64) trader;
    VarChar(8) symbol;
    Integer shares;
    Number price;
} ;
```

Is populated with 500,000 random examples created from a portfolio of 20 sample trades.

```
// From stocks.csv
@EquityTrade
trader,symbol,shares,price
NOSTRO,LU,20000,7.21
HEDGE,AOL,20000,46.99
6890459,Q,15000,22.04
and so on..
```

The trader accounts trade a random volume of shares, but the stock price is correct for each stock symbol.

7.6. Create the 67MB 500,000 Record Transaction File



Note that Kdb must be installed to run this example. See the Installation FAQ and Quick Start for information on installing Kdb.

Follow these steps to run the XML file creation.

```
// Go to the tb\cvt\Soap directory
tb -UnSoap -AnTRD
```

The console will display:

```
20000: Recv<TRD.MakeTrades2:002[UDISqlQueue/Input1]>: BEGIN EquityTrade {
    trader = <5915871>;
    symbol = <AOL>;
    shares = <100000>;
    price = <46.99>;
} END EquityTrade;
<TRD.MakeTrades2:002>::EquityTrade.passCount( 30000 ) <14,306.15/sec>: Dur: 0.70
```

This is a heartbeat log at it passes through each 10,000 records streaming to XML.

At the end of the process, the console displays:

```

          Success Bad Fld Bad Obj   Total
Totals          0      0      0      0

          Insert  Update  Voltle  Select  Total
EquityTrade      22      0      0 500000 500022
Totals           22      0      0 500000 500022

*** Processing Completed ***
SW.ON   Dur: 39.78
<DefaultThread>::Total Runtime: SW.ON   Dur: 40.06
RetCode<0>

```

In this case, the end-to-end throughput of the batch was 12,500 records per second, which includes creating the 500,000 trades and streaming them into an XML document. In the `..\output` subdirectory:

```

G:\tb\cvt\output>dir trade.xml

Volume in drive G is LOCAL
Volume Serial Number is 114D-140B
Directory of G:\tb\cvt\output

TRADE      XML      67,113,855 02-28-02  9:51a trade.xml
           1 file(s)      67,113,855 bytes
           0 dir(s)      1,422.47 MB free

G:\tb\cvt\output>

```

The very large XML file is visible. The contents of this file:

```

<?xml version="1.0"?>
<!--Generated by TierBroker 4.21-->
<!--(C) 1999-2002 Universal Data Interface Inc.-->

<TradeList>
  <EquityTrade>
    <trader>9797924</trader>
    <symbol>EMC</symbol>
    <shares>50000</shares>
    <price>23.04</price>
  </EquityTrade>
  <EquityTrade>
    <trader>4214502</trader>
    <symbol>MSFT</symbol>
    <shares>75000</shares>
    <price>65</price>
  </EquityTrade>
  <EquityTrade>
    <trader>9797927</trader>
    <symbol>IBM</symbol>
    <shares>100000</shares>
    <price>111</price>
  </EquityTrade>
  <!-- and so on... -->

```

Is lots and lots of trades.

7.7. Create a Daily P&L Report

Next, the trades will be aggregated by trader account and stock symbol.

```
// Go to the tb\cbt\Soap directory
tb -UnSoap -AnVAL
```

Each heartbeat represents 10,000 trades parsed from XML and streamed into the Kdb database.

```
30000: Recv<VAL.RevalTrades1:001[UDIXmlQueue/Input1]>: BEGIN EquityTrade {
  trader = <9797928>;
  symbol = <Q>;
  shares = <175000>;
  price = <22.04>;
} END EquityTrade;

<VAL.RevalTrades1:001>::KSQL[K]: <insert into EquityTrade price,shares,symbol,trader>
<VAL.RevalTrades1:001>::insertCache<EquityTrade[10000]> <294,117.65/sec>: Dur: 0.03
```

In this case, TierBroker is parsing records and inserting them at approximately 294,000 records per second. At the end of the process the console displays:

```
          Success Bad Fld Bad Obj   Total
Totals          0      0      0      0

          Insert  Update  Voltle  Select  Total
DailyPL          0      0      0     484    484
EquityTrade 500000      0      0      0 500000
Totals      500000      0      0     484 500484

*** Processing Completed ***
SW.ON   Dur: 01:37.22
<DefaultThread>::Total Runtime: SW.ON   Dur: 01:37.43
RetCode<0>
```

In this case, the real end-to-end through put was 5,194 records per second, which includes Internet time to complete the SOAP function calls. Throughput as high as 8,000 records per second has been observed for this process step. Results depend upon the speed of the real time stock service, the Internet connection and the size and speed of the host computer.

7.8. Daily P&L Results

The completed `dailypl.csv` tells the whole story. Each trader traded each stock roughly 1,000 times.

	A	B	C	D	E	F	G
1							
2	@DailyPL						
3	txnCount	trader	symbol	shares	mtm1	mtm2	value
471	1037	NOSTRO	DELL	80,495,000	2,040,548,250	2,051,817,550	11,269,300
472	1027	NOSTRO	EMC	84,875,000	1,955,520,000	926,835,000	-1,028,685,000
473	982	NOSTRO	GE	73,810,000	2,953,138,100	2,902,209,200	-50,928,900
474	1027	NOSTRO	IBM	79,390,000	8,812,290,000	7,814,357,700	-997,932,300
475	1079	NOSTRO	INTC	88,955,000	3,736,110,000	2,670,429,100	-1,065,680,900
476	1068	NOSTRO	JDSU	84,420,000	600,226,200	425,561,220	-174,664,980
477	1030	NOSTRO	LU	80,640,000	581,414,400	448,358,400	-133,056,000
478	1017	NOSTRO	MOT	81,535,000	966,189,750	1,074,631,300	108,441,550
479	1059	NOSTRO	MSFT	82,405,000	5,356,325,000	4,856,209,055	-500,115,945
480	1031	NOSTRO	NOK	82,750,000	1,712,925,000	1,748,507,500	35,582,500
481	1043	NOSTRO	ORCL	82,855,000	1,864,237,500	1,391,135,450	-473,102,050
482	1043	NOSTRO	PEP	85,190,000	4,097,639,000	4,308,910,200	211,271,200
483	1040	NOSTRO	PMCS	84,405,000	2,542,278,600	1,308,277,500	-1,234,001,100
484	1034	NOSTRO	Q	80,495,000	1,774,109,800	685,012,450	-1,089,097,350
485	1028	NOSTRO	SUNW	78,190,000	791,282,800	685,710,662	-105,572,138
486	998	NOSTRO	TXN	79,270,000	2,863,472,000	2,386,027,000	-277,445,000
487	1051	NOSTRO	TYC	85,435,000	4,058,162,500	2,567,321,750	-1,490,840,750
488	500000			39,072,350,000	1,265,067,073,350	1,016,992,341,197	-248,074,732,153
489							
490							

Your exact results will vary based upon real-time market conditions. Please note that past performance does not guarantee future results.

8. Edit with XML Spy

8.1. Using TierBroker with Third Party Products

TierBroker was designed to be compatible with a wide variety of third party products that support application development using open standards, such as XML and Web Services.



Readers who are primarily interested in TierBroker functionality can skip to chapter 11, *Adding an HTML Front End* to shorten this tutorial.

Third party products, such as XML Spy, make it easier to build and test TierBroker applications, but they are not a required part of the development environment.

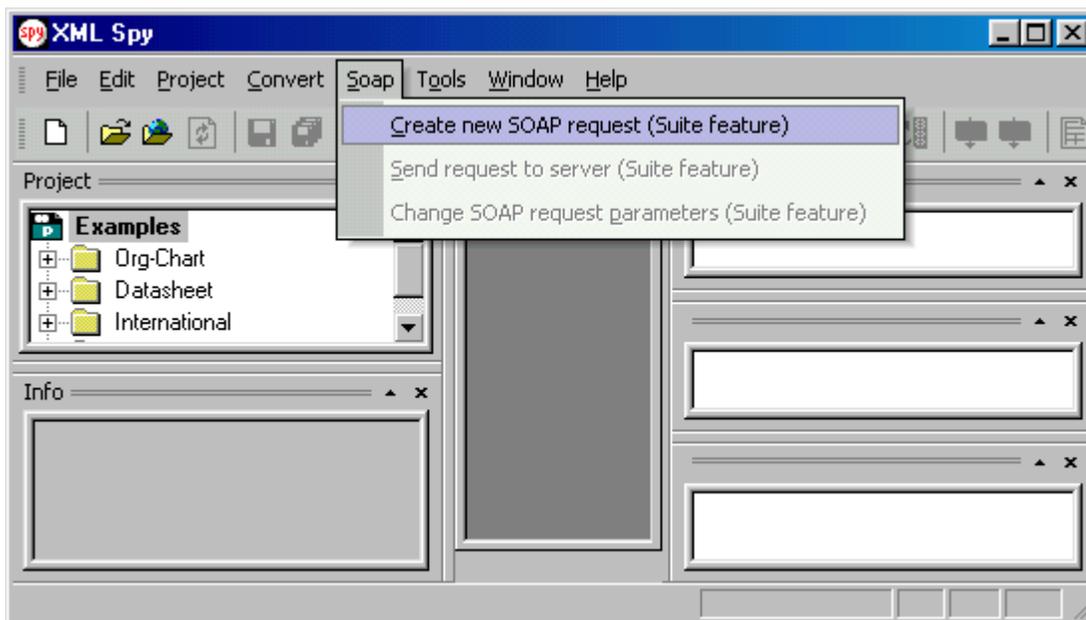
8.2. XML and WSDL Editing

XML Spy, *shaken and not stirred*, is a popular developers tool in the XML and Web Services community. It can also be used to test and validate TierBroker applications. This example assumes that the TierBroker server is still running.

```
// Go to the tb/cbt/EasySoapNow directory using the OS console
tb -UnBusinessDayCount.ts -APlocalhost:2083 -ld3
```

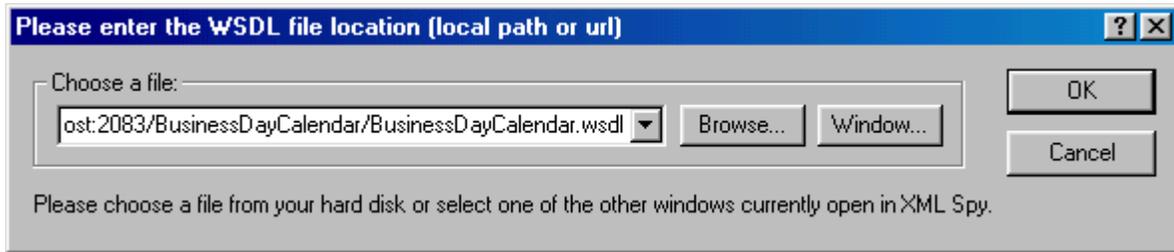
Start the XML Spy IDE application to begin this section.

8.3. Create a New Soap Request



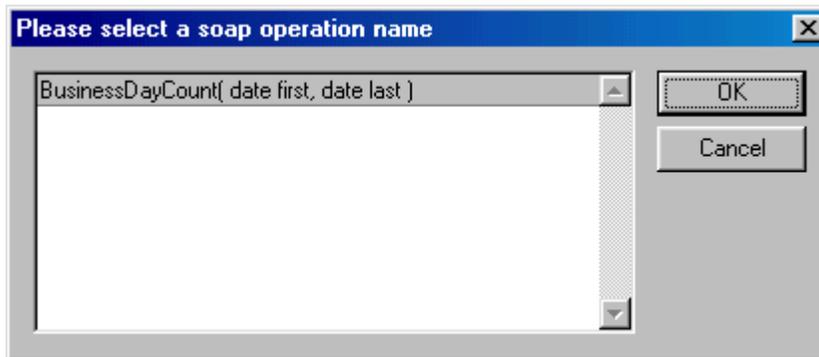
Select the Create new SOAP request from the main menu.

Then enter the WSDL URL generated by the TierBroker server.



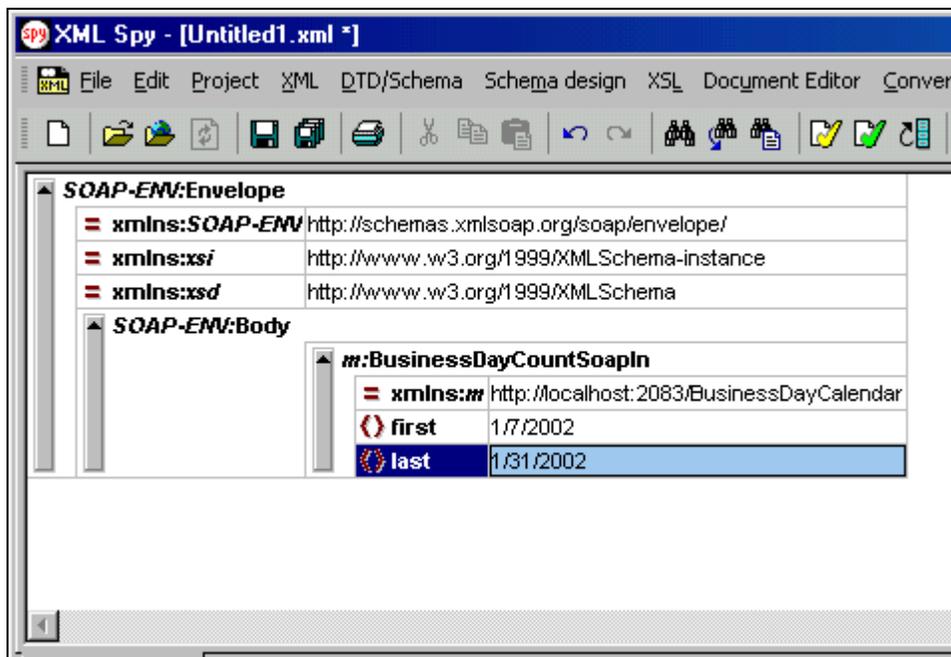
The correct WSDL file name is:

```
http://localhost:2083/BusinessDayCalendar/BusinessDayCalendar.wsdl
```



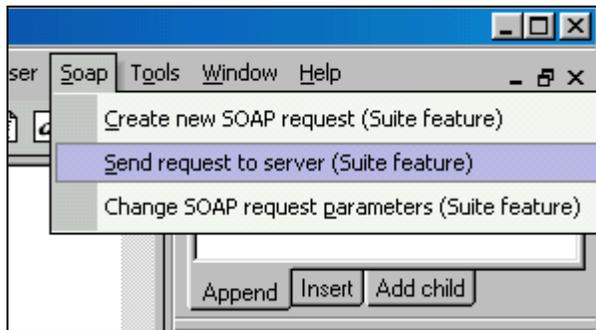
This screen indicates that XML Spy has accessed the WSDL page correctly.

8.4. Edit the Request Parameters

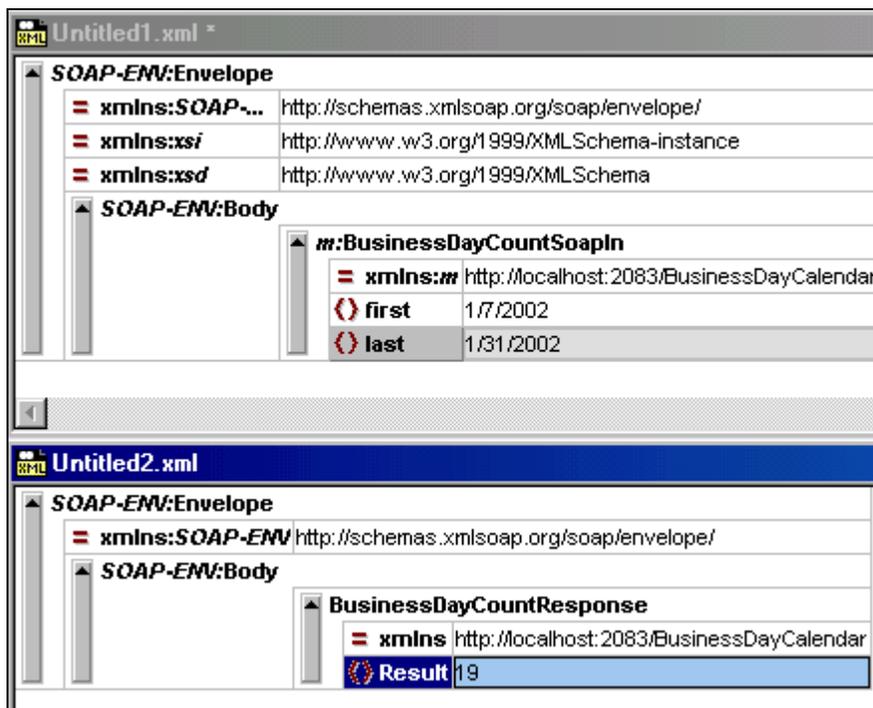


Edit the first and last date SOAP parameters.

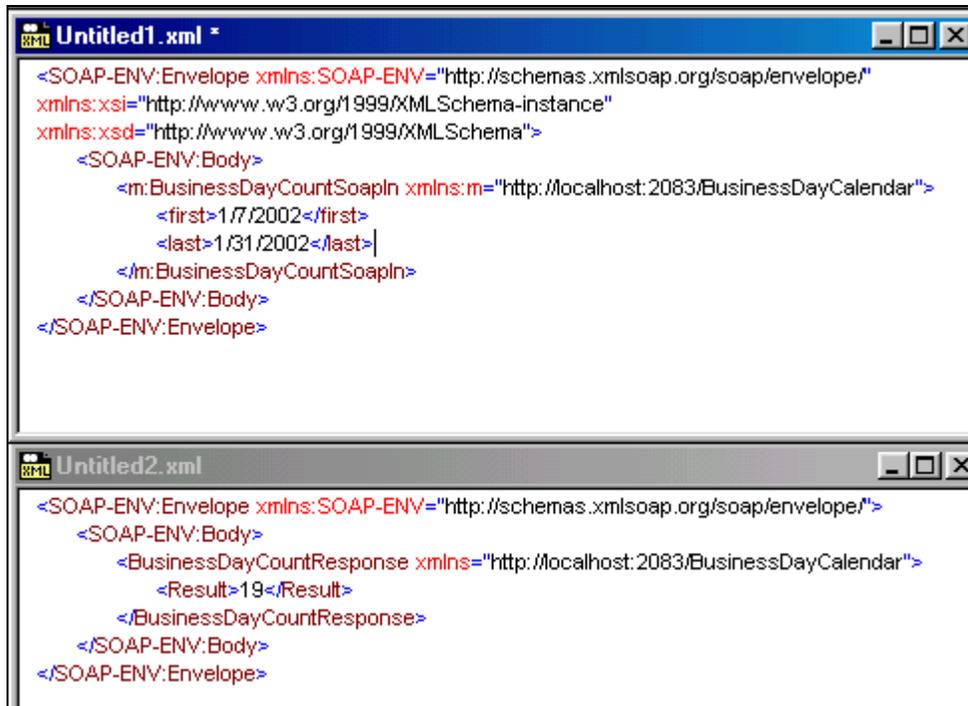
8.5. Send the Request to the TierBroker Server



Send the request to the TierBroker Server. XML Spy captures the reply.



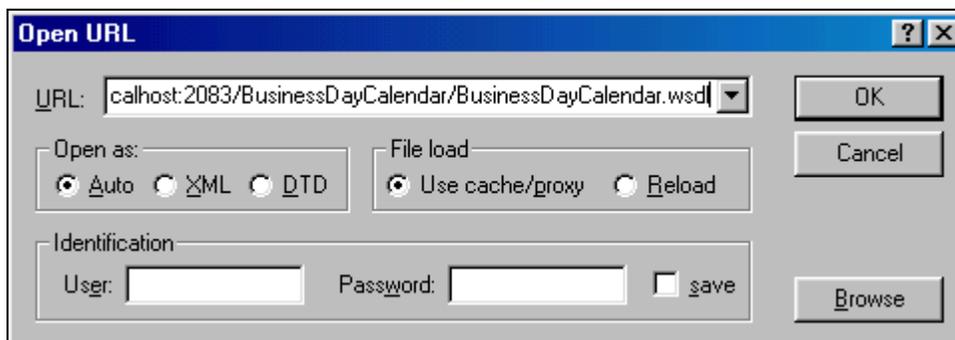
This screen shows how the XML input and output are captured side by side. In *text view*, the display shows the XML as plain text.



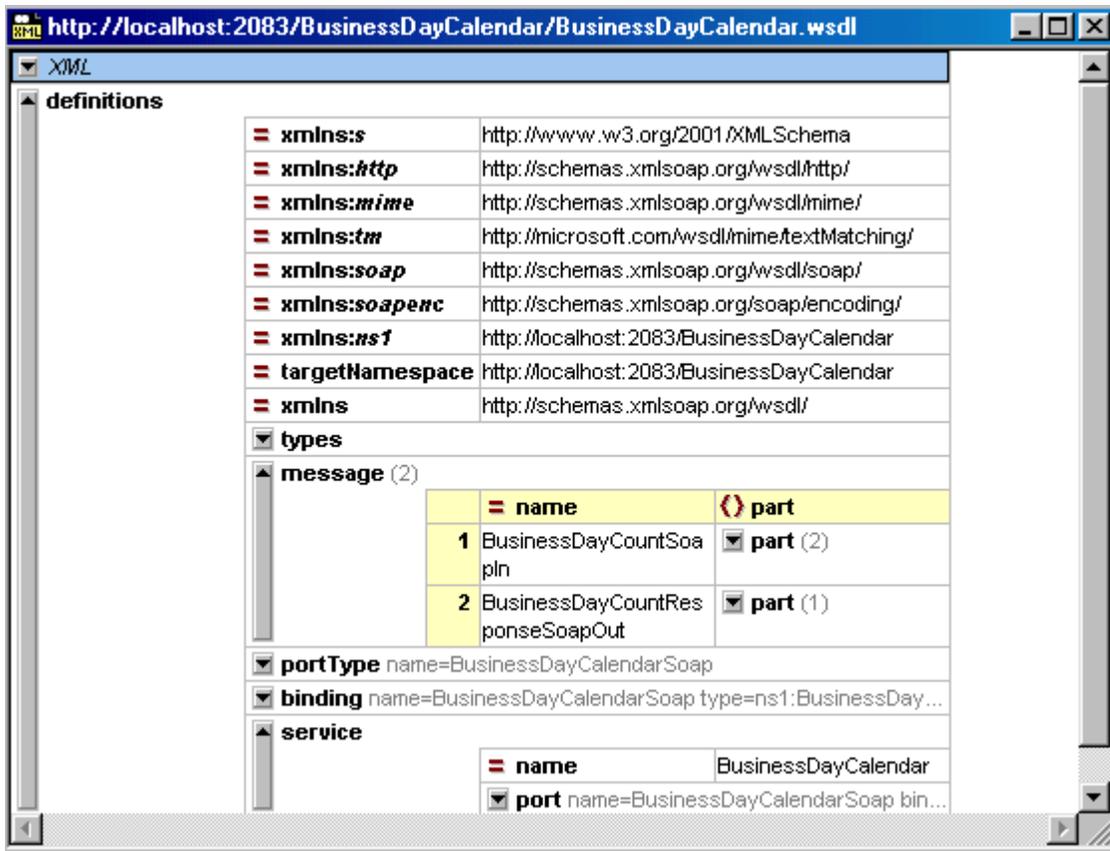
XML Spy can also be used to view or edit the WSDL.

8.6. Editing the WSDL with XML Spy

Open the WSDL page as a file.



The WSDL can be edited as an XML Spy document.



Using XML Spy with TierBroker is a good way to learn more about the mechanics of Web Services. XML Spy cannot be used to edit TBScript, since scripted logic is not XML. To edit TBScript use the TierBroker IDE or the programmer's text editor of your choice.

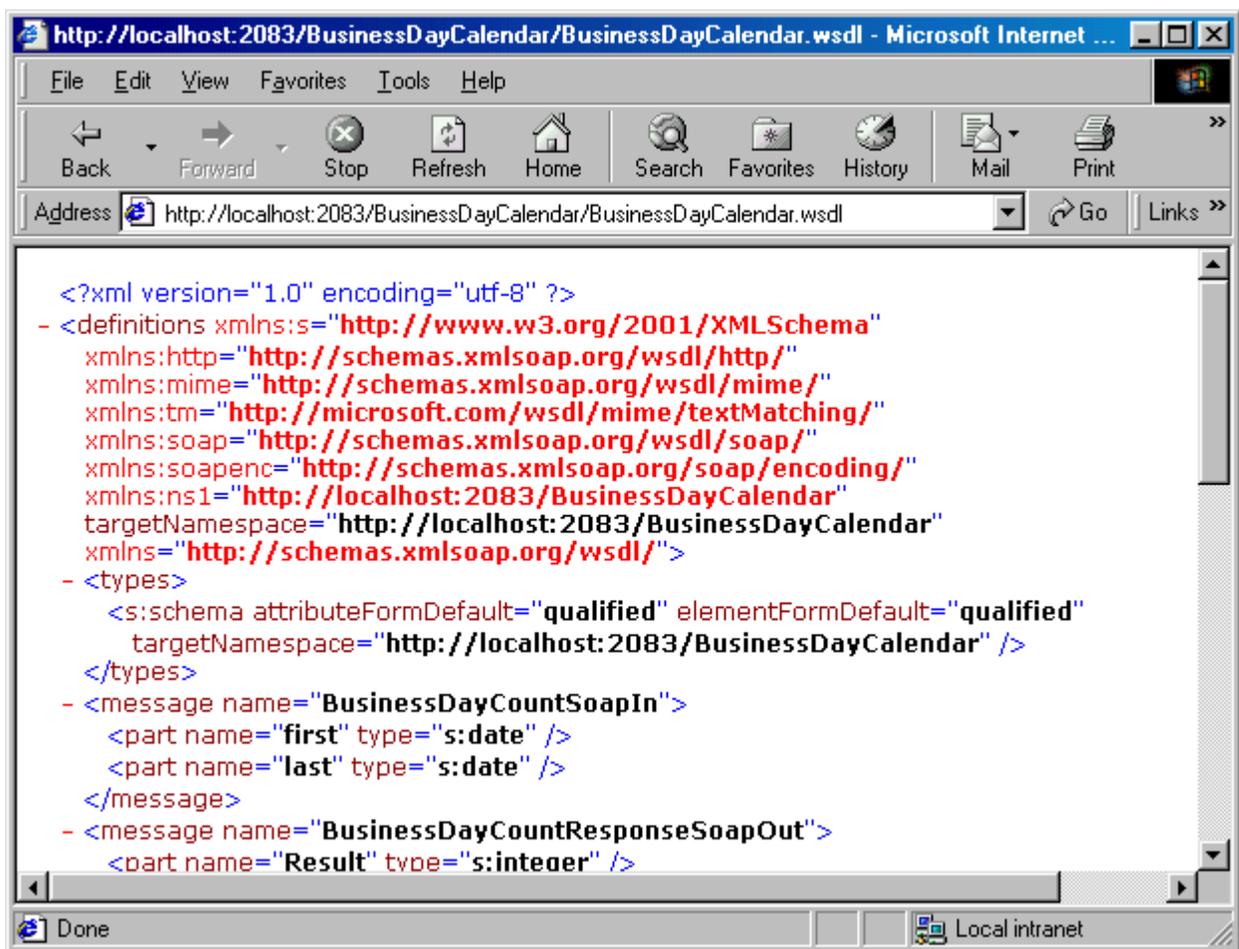
9. View the WSDL with Internet Explorer

9.1. Using IE to Access WSDL

Loading the WSDL page into IE is also a way to view and understand how TierBroker transforms functions within a script file into a WSDL service.

```
http://localhost:2083/BusinessDayCalendar/BusinessDayCalendar.wsdl
```

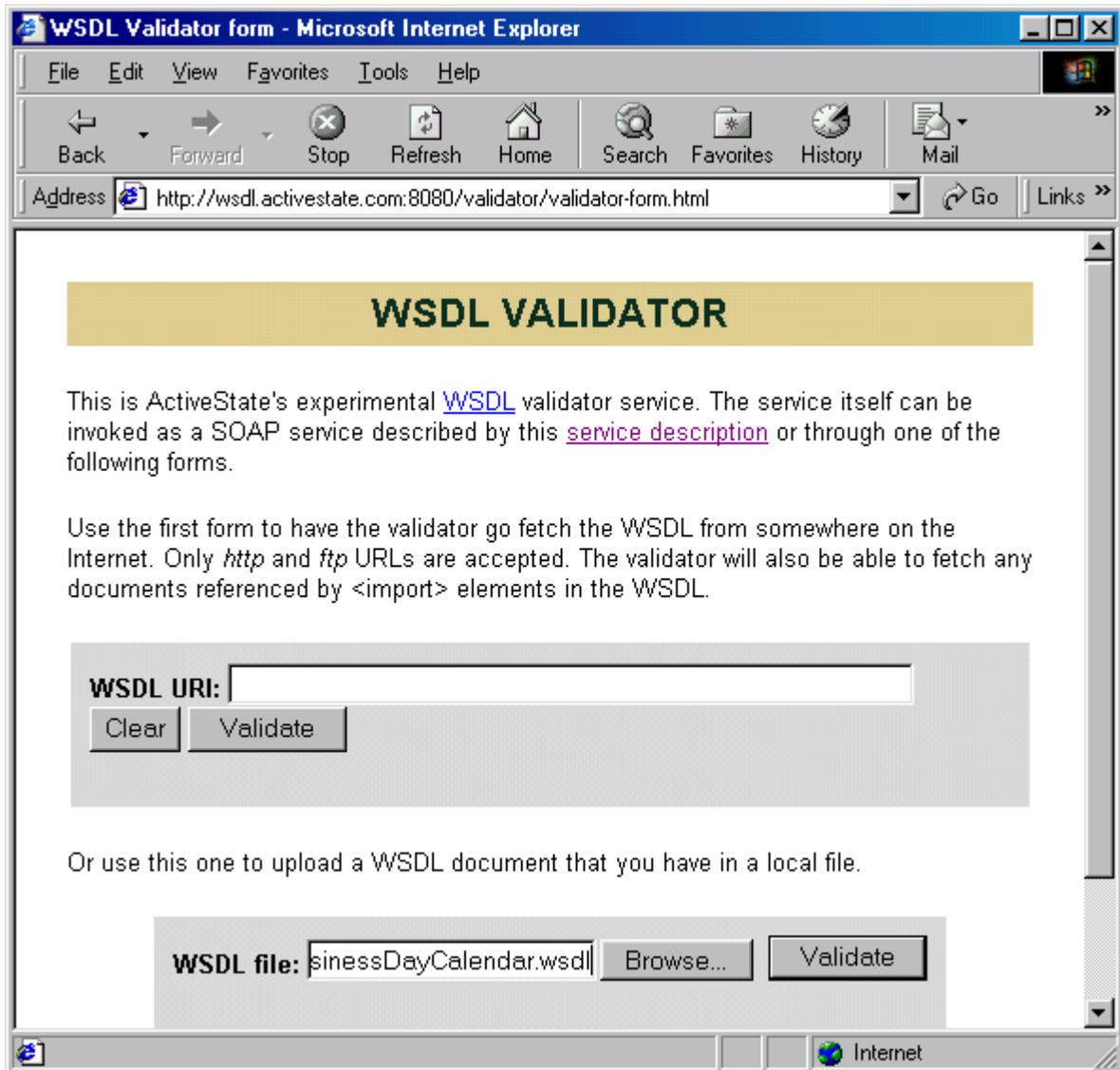
The URL of the WSDL page is the port address that is exposed by the TierBroker Server. The service name defaults to the TBScript file name and the WSDL page name defaults to the TBScript file name with a `wsdl` extension.



10. Validate the WSDL

10.1. Online WSDL Validation

ActiveState hosts a WSDL validation page. <http://wsdl.activestate.com:8080/validator/validator-form.html> can be accessed to validate a WSDL file or URL. Our experience with this service is that it is not always available to read a file, but that it is reliable if a URL is supplied instead.

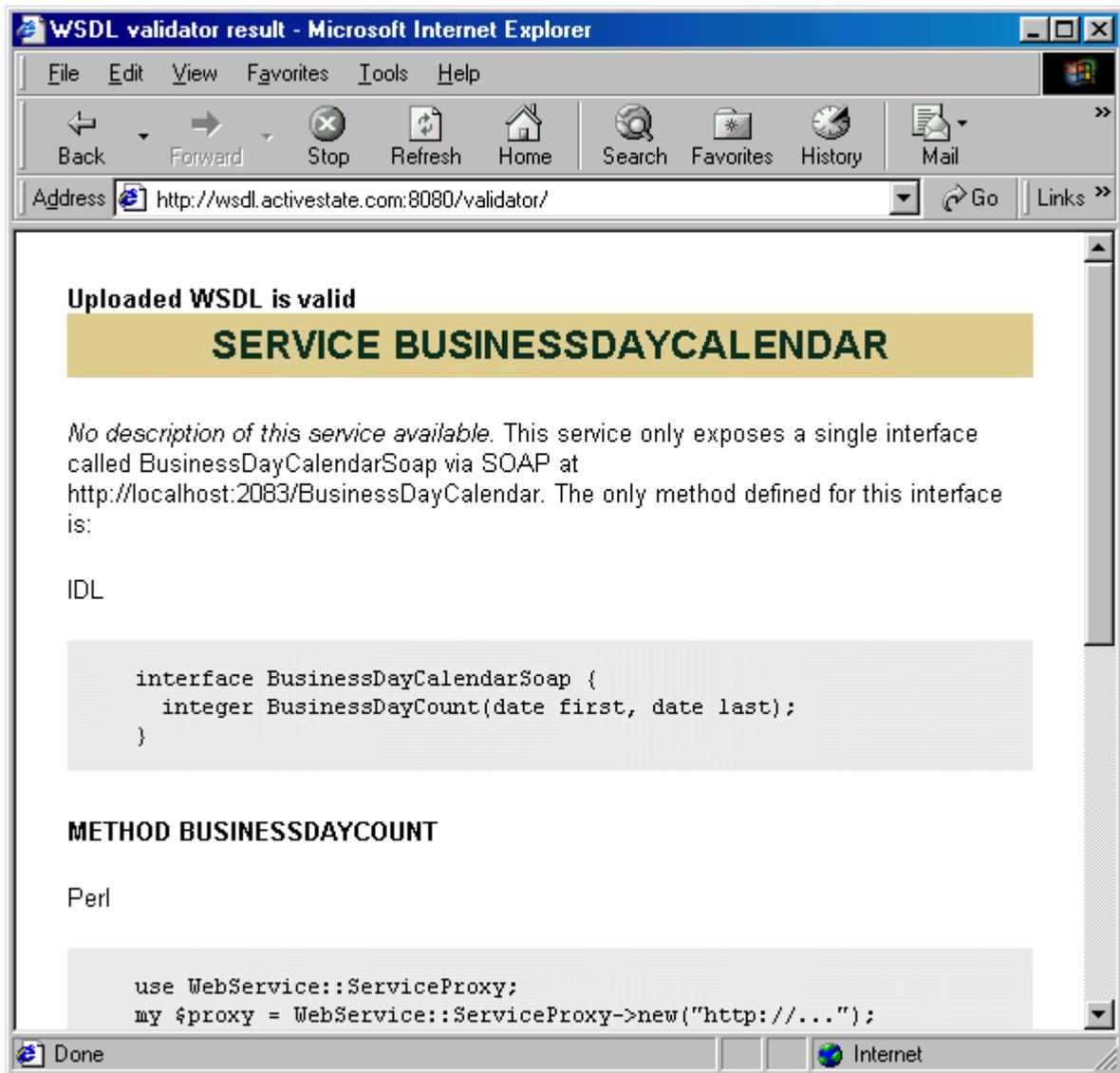


Enter the file name that TierBroker has created or use the Browse command to select the WSDL file.

```
tb\cbt\EasySoapNow\BusinessDayCalendar.wsdl
```

If TierBroker is running on an open port, then the hostname can be used instead of the text file name.

10.2. Successful Validation



The TierBroker WSDL file is valid and can be used with a variety of SOAP clients.

11. Trace the HTTP

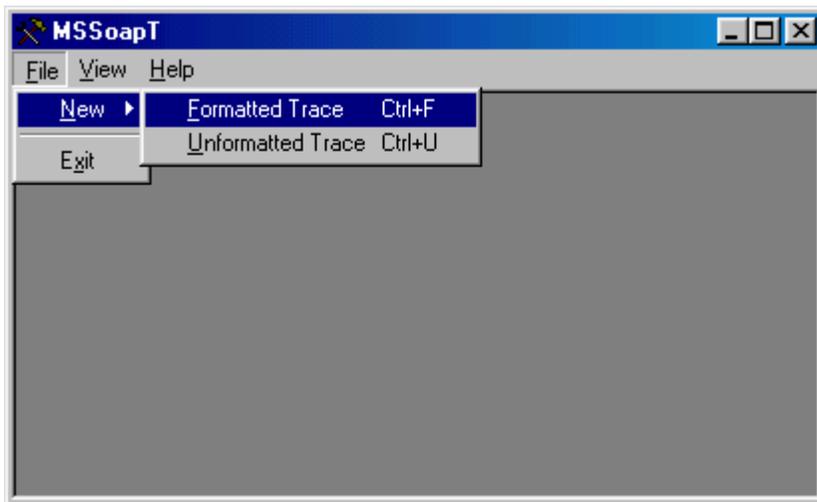
11.1. Using the MS Soap Toolkit

The MS Soap toolkit has a TCP/IP trace facility. This utility can be used to view the XML transferred between the client and server applications.

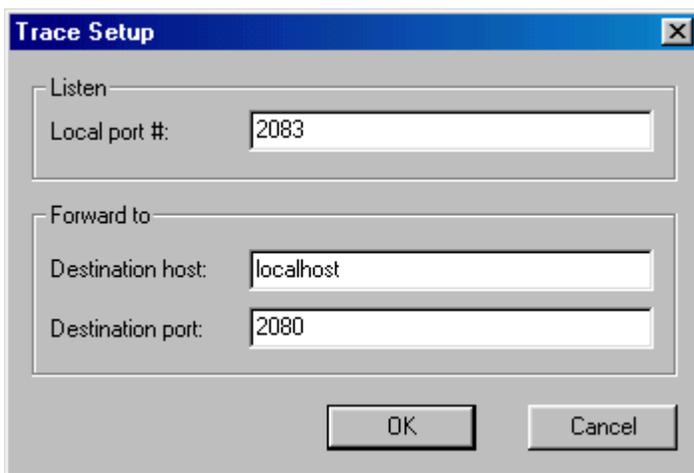
It is not necessary to understand HTTP logging in order to use TierBroker. This example is included for developers who wish to learn more about advanced low-level debugging techniques.

The TCP/IP trace utility is useful in cases where ordinary TierBroker logging is insufficient to diagnose and correct a processing error occurring with a remote host.

11.2. Open a Formatted Trace



First select Formatted Trace from the main menu.



Redirect port 2083 to port 2080.

11.3. Run the TB Server

From the command line:

```
tb -UnBusinessDayCalendar.ts -APllocalhost:2080
```

This starts the server on port 2080. It will receive data from port 2083, which is being redirected.

11.4. Run a Second TB Server as a Client

From a second command line:

```
tb @tb2 -UnBusinessDayCalendar.wsdl
```

This starts the TBSOap application, and loads the WSDL file that describes the BusinessDayCalendar Web Service.

Execute a Soap Request

Execute a request to the first server:

```
tb>soap BusinessDayCalendarSoap.BusinessDayCount, 1/7/2002, 1/31/2002
```

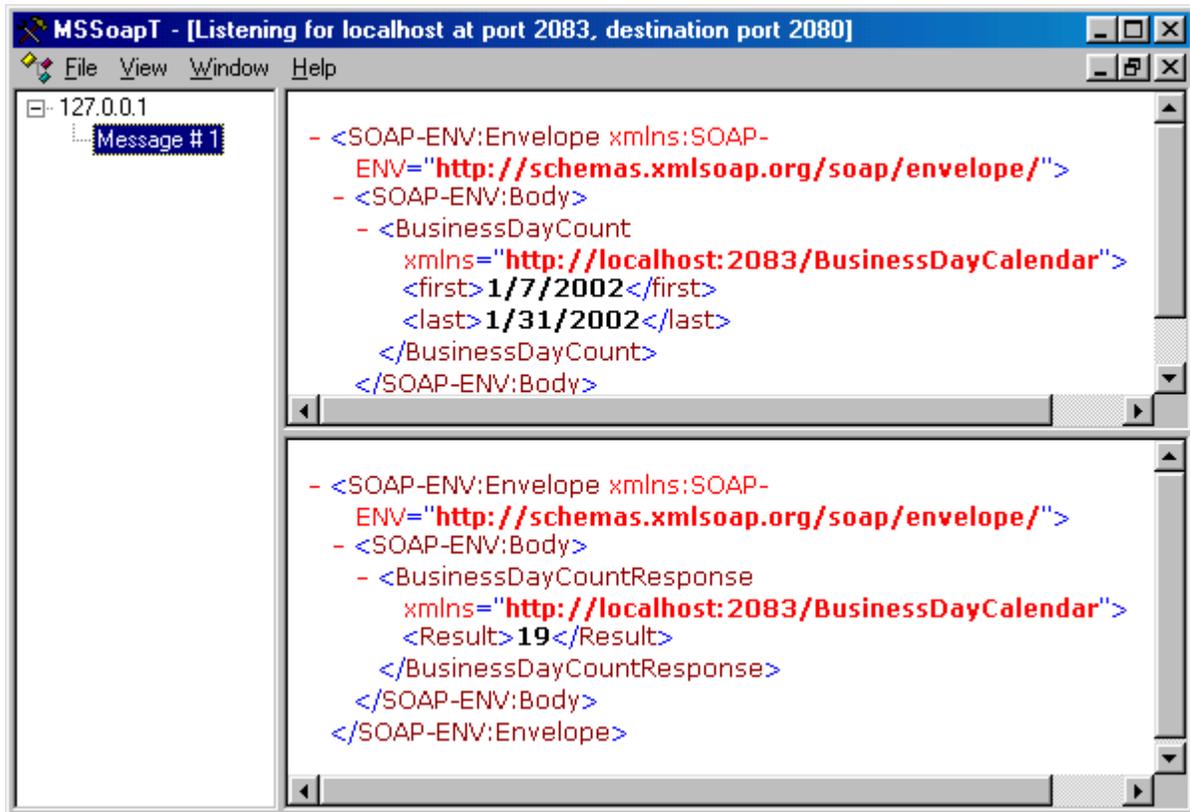
The server console shows the command results:

```
<TBRequestStr001>::SOAP[BusinessDayCalendarSoap.BusinessDayCount]: <Dur='44.12'  
Req='1/7/2002, 1/31/2002' Resp='19'>
```

TierBroker can be used to test all types of Web Services. *See also* `tb/cbt/Soap` for a listing of Web Services from `xmethods.org`, which have been tested with TierBroker.

11.5. View the Trace Results

In the MS Soap Trace click on the first transaction and view the results in XML.



12. Adding an HTML Front End

12.1. Create an HTML Page for BusinessDayCount()

```

<!-- (C) Universal Data Interface Corporation 2001-2002 -->
<HTML>
<HEAD>
  <TITLE>Business Day Calendar</TITLE>
</HEAD>
<BODY>
  <FORM method='post' action='BusinessDayCalendar/BusinessDayCount'>

  <H1><FONT color='red'>Business Day Count</FONT></H1>

  <BR/>First Day:
  <INPUT type='text' name='first' value='1/7/2002' size='30' />

  <BR/>Last Day:
  <INPUT type='text' name='last' value='1/31/2002' size='30' />

  <BR/>
  <INPUT type='submit' value="Submit" />
  </FORM>
</BODY>
</HTML>

```

All it takes to access BusinessDayCount () from an HTML page is to make the ACTION of the FORM POST the SOAP Action of the WSDL page.

12.2. Files Used in this Example

```

BusinessDayCalendar.ts // TBScript source code
BusinessDayCalendar.html // HTML page

```

12.3. Run the Server

Execute the TierBroker Server.

```

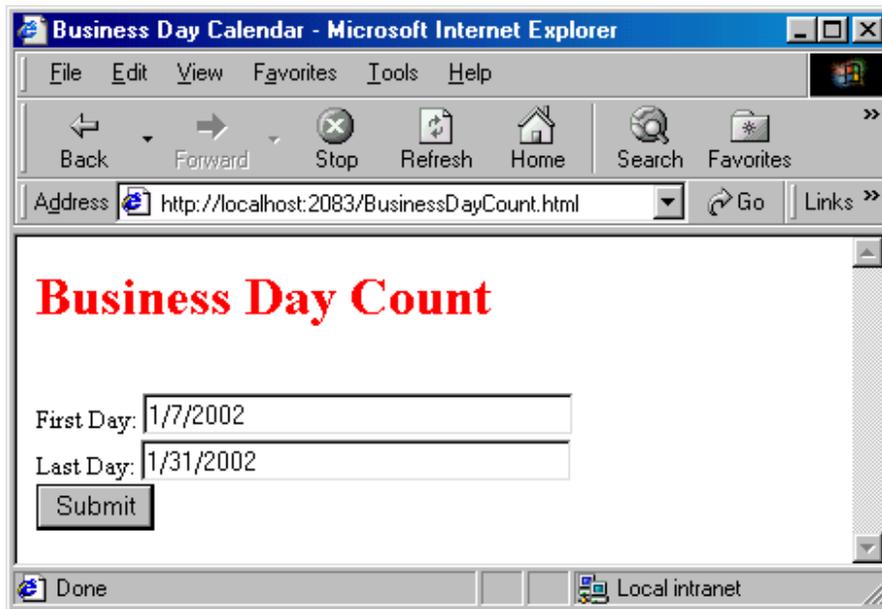
// Go to the tb/cbt/EasySoapNow directory
tb -UnBusinessDayCalendar.ts -APllocalhost:2083 -ld3

```

Now TierBroker is waiting for SOAP requests.

12.4. Use the HTML Form

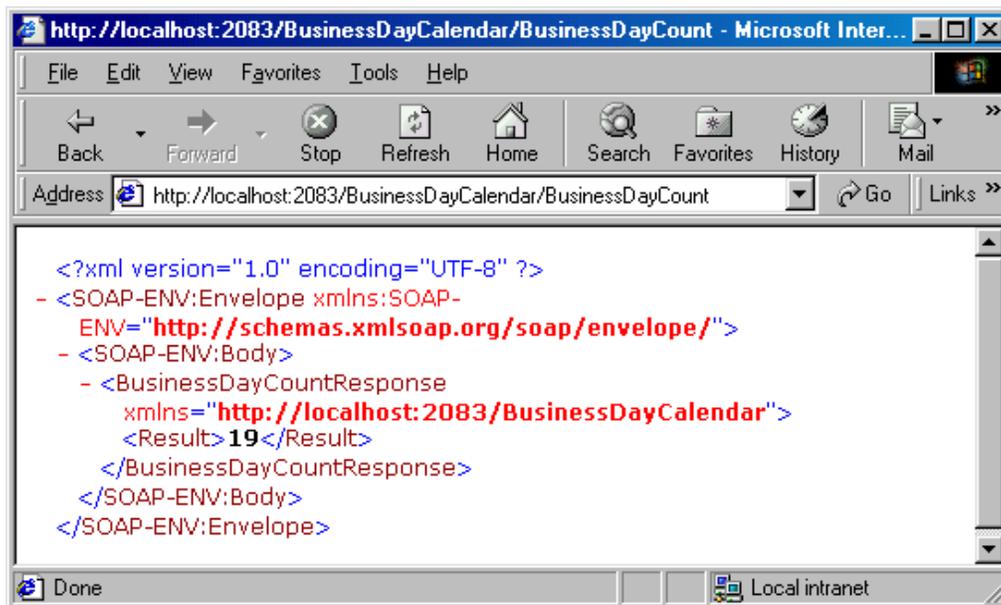
Enter the TierBroker Server URL `http://localhost:2083/BusinessDayCount.html`.



Press the Submit button to use this form.

12.5. View the SOAP Result

TierBroker returns the SOAP result.



This is a great start. TierBroker makes it easy to call a Web Service from a Browser page. In the next example, TierBroker is used a proxy to a external Web Service that is not HTTP Get/Post enabled, and renders that result into an HTML page.

13. HTML Proxy for SOAP Services

13.1. Calling External Web Services from HTML

TierBroker can be used to call external Web Services that are not written in TBScript.

☞ Even if a SOAP service does not define an HTTP GET/POST port, TierBroker can enable that capability by acting as a proxy.

In this case TierBroker can run as a plug-in on the client side, or in the data center on the server side. The target of the FORM ACTION defines the location of the TierBroker Server.

13.2. Primary Files Used in this Example

The following files are actors in this scenario:

```
StockQuote.udi           // The TierBroker project source code
StockQuote.html         // The HTML input form
StockQuoteResponse.html // The HTML response template
StockDisclaimer.html    // Stock quote disclaimer text
```

These files summarize the entire work effort required to produce this application.

13.3. Other Files Used in the StockQuote Application

The StockQuote application demonstrates three SOAP response rendering techniques – templates, XSLT and functions. The easiest approach uses TierBroker templates, which are plain-text HTML. The following files are used to demonstrate XSLT and scripting as methods used to render the SOAP response.

```
StockQuote-XSL.html     // HTML page using XSLT rendering
StockQuote.xsl          // Stock Quote XSLT style sheet
StockQuote.ts           // Source code for TBScript rendering
StockQuote-Fxn.html    // HTML page for TBScript rendering
```

These files are used in examples that follow this chapter.

13.4. To Run the StockQuote Application

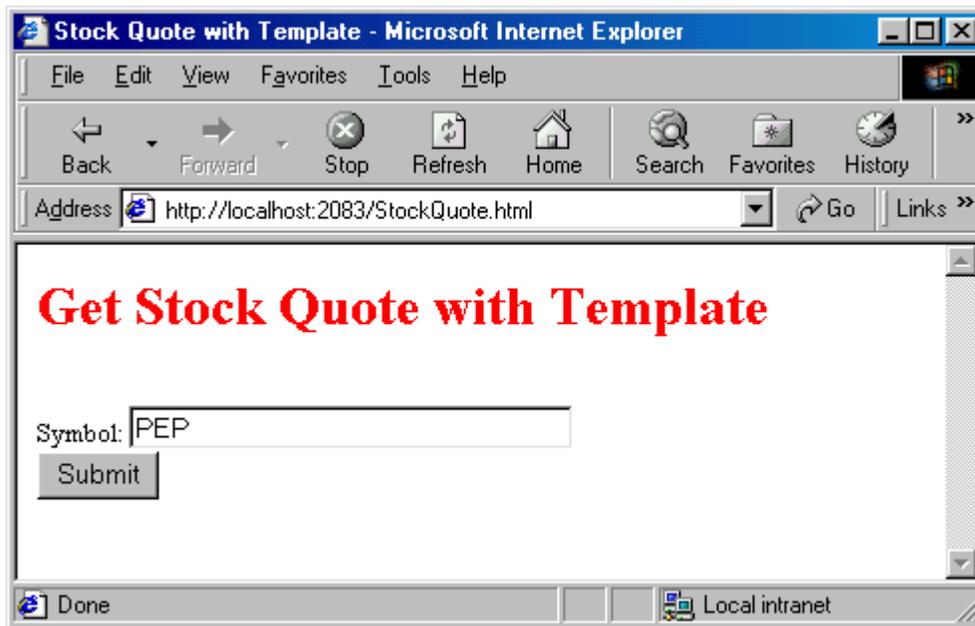
First run TierBroker from the operating system console:

```
cd tb\cbt\EasySoapNow
tb -UnStockQuote.udi -APlocalhost:2083 -ld3
```

Then access the input form with the Browser.

```
http://localhost:2083/StockQuote.html
```

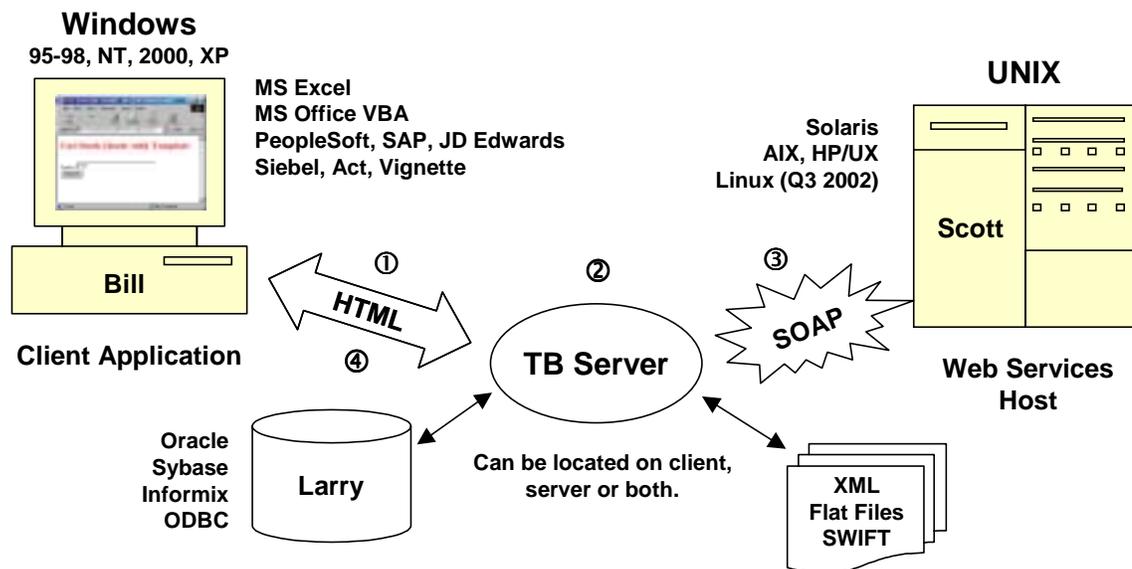
The Browser should display the following form:



Note that you must have an Internet connection to access the external stock quote price service.

13.5. Understanding the StockQuote Project

This project uses an HTML input form to call TierBroker as a proxy for an external web service.



1. This HTML form calls TierBroker as the ACTION of the HTML POST. The ACTION contains the name of the SOAP Action to be invoked on the external server.
2. TierBroker renders the POST into a SOAP request and calls the external service.

3. TierBroker receives the SOAP response and renders it into HTML using a template. XSLT is used as an alternative in another section.
4. The HTML response is delivered back to the Browser.

TierBroker can proxy this transaction on either the client side or the server side. The advantage of a Client Side Deployment is that TierBroker can be used to extract or insert information into client side applications. The advantage of server side deployment is that all the client requires is a Browser to access the application.

13.6. The StockQuote Web Service

The WSDL page:

```
http://services.xmethods.net/soap/urn:xmethods-delayed-quotes.wsdl
```

This service defines a simple API that takes a stock symbol and returns the stock price.

13.7. HTML Page with Proxy Call

The following HTML page:

```
tb/cbt/EasySoapNow/StockQuote.html
```

is the Browser entry point for this example.

```
<!-- (C) Universal Data Interface Corporation 2001-2002 -->
<HTML>
<HEAD>
  <TITLE>Stock Quote with Template</TITLE>
</HEAD>
<BODY>
  <FORM method='get' action='GetStockQuote$'>

  <!-- Select Response Format Template -->
  <INPUT
    type  ='hidden'
    name  ='udi:template'
    value ='StockQuoteResponse.html' />

  <H1><FONT color='red'>Get Stock Quote with Template</FONT></H1>

  <BR/>Symbol:
  <INPUT type='text' name='symbol' value='PEP' size='30' />

  <BR/>
  <INPUT type='submit' value="Submit" />
  </FORM>
</BODY>
</HTML>
```

In this case, the SOAP Action is aliased with the TierBroker project variable `GetStockQuote$`. This is done for two reasons:

1. To allow the HTTP end point to be switched to a `localhost`, when the Internet is not available for testing or demonstration.

2. To mask characters that occur in the real HTTP end point, such as '#', which are not permitted in an HTTP action⁵.

Allowing TierBroker to be a proxy also gives us the opportunity to enrich or modify the request.

☞ When TierBroker is used as a proxy, there does not have to be a one-to-one correspondence between the HTML form and the Web Service API.

The information regarding how TierBroker will proxy the SOAP request is stored in a project file.

13.8. *Creating the TierBroker Project File*

The following XML document is the project file that the TierBroker Server reads at runtime.

```
<Project id = 'Easy SOAP Now'
  author    = '(C) Universal Data Interface Corporation 2002'>

  <Script>
  #include 'BusinessDayCalendar.ts'
  #include 'StockQuote.ts'
  </Script>

  <VarList>
    // Need this alias, because '#' will not go through HTTP
    GetStockQuote$ = 'urn:xmethods-delayed-quotes#getQuote'
  </VarList>

  <SoapRequestList>

  <SoapRequest
    wsd1='http://services.xmethods.net/soap/urn:xmethods-delayed-quotes.wsdl' />

  </SoapRequestList>
</Project>
```

One line aliases the SOAP Action, and another line loads the WSDL page.

13.9. *Rendering the SOAP Response into HTML*

In the next section the SOAP response is rendered into HTML.

⁵ The # could be aliased with %23, and this is also a valid approach.

14. HTML Rendering with Templates

14.1. Rendering the StockQuote Response

This section continues the Stock Quote example application.

14.2. Rendering Methods

TierBroker support three rendering methods – Templates, External Processors and Scripted Functions. The most common external processor is XSLT. These methods can also be combined – for example, a scripted function can be used to enrich or validate the SOAP response before it is rendered with XSLT.

14.3. TierBroker Templates

TierBroker templates are nothing more than HTML documents. Note that TierBroker parses both HTML and XHTML indifferently.

☞ TierBroker templates are HTML documents with the addition of the TB element. The HTML can be generated from programs like *FrontPage* and *DreamWeaver*. If the Browser can read it, so can TierBroker!

The TierBroker TB element allows information from the SOAP request and response to be combined in the HTML result.

The Author's Experience with XSLT

The choice of using templates or XSLT is a matter of preference. At UDICo, XSLT is used to render the XML Server Documentation into the on-line HTML help files. The biggest problem I have visualizing XSLT results is that templates are functional, and you need to understand the order of the processing steps in order to predict the structure of the finished document. The primary design goal of templates was to create a less general transformation that could be viewed in a Browser or HTML editor as part of a WYSIWYG process. I wanted to make the process easier and more familiar for web developers. – Adam Greissman

Advantages of TierBroker Templates Over XSLT

There are three principle advantages to using templates.

- Templates are HTML documents that can be viewed in the Browser or an HTML editor. This is a WYSIWYG design approach.
- Templates have access to both the SOAP request and the SOAP response. XSLT can process only one input document at the present time.
- XSLT requires an external processor, which may have a performance impact for large documents or high volume applications.

14.4. Indicating Rendering Steps in HTML Forms

The HTML input form contains a hidden INPUT control that indicates a processing directive to the TierBroker Server.

```
<BODY>
  <FORM method='get' action='GetStockQuote$'>

  <!-- Select Response Format Template -->
  <INPUT
    type  ='hidden'
    name  ='udi:template'
    value ='StockQuoteResponse.html' />

  <H1><FONT color='red'>Get Stock Quote with Template</FONT></H1>

  <BR/>Symbol:
  <INPUT type='text' name='symbol' value='PEP' size='30' />

  <BR/>
  <INPUT type='submit' value="Submit" />
  </FORM>
</BODY>
```

In this case, the processing directive is `udi:template` and the name of the template is `StockQuoteResponse.html`. Other directives include `udi:system` and `udi:function`, which are used in subsequent examples.

14.5. The StockQuote Response Template

Cascading Style Sheet Header

The first section of the response template is an ordinary cascading style sheet.

```
<!-- (C) Universal Data Interface Corporation 2002 -->

<HTML>
  <HEAD>
    <TITLE>Real Time Stock Quote</TITLE>
    <STYLE>
      BODY { FONT: 10pt verdana }
      SELECT { FONT: 10pt verdana }
      TD { FONT: 10pt arial }
      INPUT { FONT: 10pt verdana }
      TH { COLOR: black; FONT: bold 20pt verdana }
      .RH1 { COLOR: green; TEXT-ALIGN: center; FONT: 20pt arial black; }
      .RH2 { COLOR: black; TEXT-ALIGN: center; FONT: 28pt times roman; WIDTH: 150; }
      .RH3 { COLOR: red; TEXT-ALIGN: center; FONT: 20pt arial black; WIDTH: 150; }
      .SQ1 { COLOR: black; TEXT-ALIGN: center; FONT: 32pt verdana bold; WIDTH: 150; }
      .FP1 { COLOR: black; TEXT-ALIGN: left; FONT: 8pt times roman; WIDTH: 450; }
      .FP2 { COLOR: black; TEXT-ALIGN: left; FONT: 8pt arial black; }
      .IMG { TEXT-ALIGN: center; }
    </STYLE>
  </HEAD>
```

This is standard HTML, so it is not reviewed as a part of this tutorial.

HTML Body

The HTML BODY contains a simple table, which as three <TB> elements.

```
<BODY>
<TABLE border='0' cellPadding='2' cellSpacing='1'>

  <TR><TH colspan='3'>Call Your Broker Quick $$$</TH></TR>

  <TR>
  <TD class='RH1'>Buy Low!</TD>
  <TD class='RH2'><TB request='Body/getQuote/symbol' /></TD>
  <TD class='RH3'>Sell High!</TD>
  </TR>

  <TR>
  <TD class='IMG'><img border='0' src='bull.jpg' /></TD>
  <TD class='SQ1'><TB response='Body/getQuoteResponse/Result' /></TD>
  <TD class='IMG'><img border='0' src='bear.jpg' /></TD>
  </TR>

  <TR><TD colspan='3' class='FP1'>
  <TB href='StockDisclaimer.html' />
  </TD></TR>

</TABLE>
</BODY>
</HTML>
```

<TB> Attributes request|response|href

There are three supported attributes supported by the <TB> element:

request – Select material from the SOAP request using an XPATH statement.

response – Select material from the SOAP response using an XPATH statement.

href – Include a file or URI for processing, which in turn can be another template.

See the *TierBroker Reference Manual* for a complete review of how XPATH is implemented in TierBroker.

14.6. The StockQuote HTML Result



The HTML result combines graphics as well as text. All that is required to produce this page is plain-text HTML. No ASP, JSP, VB or .NET programming is required. More importantly, TierBroker can be used in a Windows NT environment with IIS as easily as it can be used with .NET or J2EE systems running WebSphere or iPlanet.

The following sections look at how to process complex forms as input, and how to create tables as output using TierBroker templates.

14.7. Advanced Formatting with the TB Element

The following material is optional.



Skip to the next chapter to continue this tutorial with *XSLT and External Processors* to complete the basic version of this tutorial.

TierBroker supports fine grained control for rendering and localizing Browser applications. This knowledge is not necessary for building basic applications.

14.8. Debugging HTML Applications

When viewing complex forms, Internet Explorer will sometimes return this red X's where JPG images belong. This is because the images are cached, but the Browser cache has become out of synch with the HTML page. The TierBroker application never receives the GET request for those graphics.

Select Tools-Internet Options-Delete Files to delete cached graphics files and force IE to read from the server.

HTTP logging can also be enabled for the TierBroker server with the `-ld3` command line parameter, when the TierBroker Server is started. If there is no HTTP request for a file, that indicates the Browser cache should be refreshed.

14.9. Using the TEXT Element

The TEXT element is not a part of HTML, but can be used to enclose printed text.

```
<HTML><BR><TEXT>Stock Price: </TEXT><TB response='+value' /></HTML>
```

This is useful when concatenating text fields to TB elements. Note that TEXT elements are not trimmed or altered for whitespace when they are parsed by TierBroker – TEXT is delivered to the Browser *as is*.

14.10. Including Sample Text

Sample text is useful in templates, because the TB element is substituted at runtime and is not visible at HTML design time.

```
<TEXT>Stock Price: </TEXT><TB response='+value' /><TEXT udi:sample='1'>123.0</TEXT>
```

The `udi:sample` attribute indicates that this markup is *sample text* and should not be used at runtime.

☞ The `udi:sample` attribute indicates that an element and all of its children should be omitted from the rendered HTML page.

In the Browser this fragment displays as:

```
Stock Price: 123.0
```

...which is useful for visualizing how this template will appear when it renders a SOAP response at runtime. If the actual stock price is 80.01, then at runtime the sample text will not be printed and the markup will be rendered as:

```
Stock Price: 80.01
```

Note that any HTML element can use the `udi:sample` attribute to indicate that it should not be processed. In that case that element and all of its children are omitted from the completed HTML page.

14.11. Formatting Numbers, Dates and Lists

The `request` and `response` values may also be modified with formatting parameters, that effectively bind them as specific data types.

```
<history>
  <price>12345</price>
  <date>3/5/2000 12:14:00</date>
  <agentList>
    <agent>Bob</agent>
    <agent>Carol</agent>
    <agent>Ted</agent>
    <agent>Alice</agent>
  </agentList>
</history>
```

XML does not always have the formatting required for it to be presented on a Web Page. In this example, the `price`, `date` and `agentList` will be reformatted.

14.12. Formatting Numbers

The following TB element reformats the `price` element content.

```
<TB response='price' numberfmt="'',',','.',2,1"/>
```

The format is a comma delimited list. Each item in the list must be single or double quoted. This example of the `numberfmt` attribute of the TB element sets the following values:

Thousands Grouping The '000's' will be grouped with a comma.

Decimal Point The decimal separator will be a period.

Precision There will be two values after the decimal point. Zeros will be used to right-pad the string if the number rounds to a whole number value.

Divisor In many reporting applications values are reported in the thousands. The divisor specifies a floating point value what will be used as a divisor. The default value is 1 . 0.

In the case given above, the price '12345' will appear in the HTML as 12 , 345 . 00.

14.13. Formatting Dates

Dates represent a challenging case for parsing and formatting. A complete explanation of date formatting is contained in the on-line TierBroker Server documentation under the *UDIModel Enumerations and Classes* `tb\bin\cmd\english\FormatSet.html` web page.

The following TB element reformats the `price` element content.

```
<TB response = 'date' datefmt = "DDMMYYYY, '- '"/>
```

In this case, the date entry will be parsed as a date using the global default date format. The default value is `DDMMYYYY`, but it can be modified by creating a `FormatSet` entry with an id of `default` in the TierBroker project file. Note that in this case the dash – character is indicated as the separator character.

```
<date>3/5/2000</date> // This value is change to..
03-May-2000 // Using the format given above
```

The `datetime` attribute can be used to render the content as a `Datetime` data type.

```
<TB response = 'date' datetimefmt = "MMDDYYYY"/>
05/03/2000 12:15:00
```

In this case, the timestamp portion of the XML element content is also rendered.

Localizing Day and Month Names

The names of months, short months and days of the week may also be localized with the following TierBroker project variables.

LongMonth\$ List of 12 months starting with January.

ShortMonth\$ List of 12 months starting with Jan.

LongDay\$ List of 7 days starting with Monday.

ShortDay\$ List of 7 days starting with Mon.

To run more than one set of local language definitions for number and date formatting parameters it may be necessary to run a separate TierBroker server instance for each set of format parameters.

14.14. Indicating the Source Date Format

In some cases, the *source* date format is not the same as the TierBroker default date format, and both the *source date format* and the *target date format* must be specified.

```
<TB response = 'sourcePath' datefmt = 'sourceFormat;targetFormat' />
```

The *source format* is separated from the *target format* by a semicolon. For example:

```
<TB response='return/dep_date' datefmt = 'YYYYMMDD;DDMMYY, - ' /></TD>
```

The `datefmt` attributes gives the *source date format* as `YYYYMMDD` and the *target date format* as `DDMMYY` (separated by a semicolon) for the HTML template.

14.15. Formatting Lists

Lists can be formatted using a delimiter. Note that this is not the same idea as formatting HTML tables, which is covered in a separate example.

```
<agentList>
  <agent>Bob</agent>
  <agent>Carol</agent>
  <agent>Ted</agent>
  <agent>Alice</agent>
</agentList>
```

In this case, the list of agents can be extracted as a single list.

```
<TB response = 'agentList/agent' delimfmt = ", " />
```

The `delimfmt` attribute extracts the matching `agent` elements as:

```
Bob,Carol,Ted,Alice
```

Note that only a single text character can be used as the delimiter.

14.16. Setting HTML Attribute Values with the TB Element

The `TB` element provides a flexible method for extracting information from the SOAP request and response documents. In some cases this information will be used to set hidden information in controls, such as session number, transaction code or other session state information.

```
<!-- Does not work, because it is not XML -->
<INPUT type='hidden' name="sessionId' value="<TB response='sessionId' />" />
```

In this case, the `udiatt` namespace attribute can be used to indicate that the result of the child nodes should be extracted to form the attribute value.

```
<!-- Rendering engine converts this at transform time -->
<INPUT type='hidden' name="sessionId'
  <udiatt:value><TB response='sessionId' /></udiatt:value>
</INPUT>
```

This `INPUT` will be converted into a single HTML element when it is transformed from an XHTML template into an HTML document.

```
<!-- HTML result -->
<INPUT type='hidden' name="sessionId' value='12345'>
```

Note that the `INPUT` control element is not closed with `" />"` when it is rendered to HTML. This is true for all HTML 4.0 empty elements.

15. XSLT and External Processors

15.1. Using XSLT to Render HTML

This section continues the Stock Quote example application and uses an external XSLT processor to render the SOAP response into an HTML document. Note that any external processor can be used with TierBroker to render the HTML response.

 To run this example the Saxon XSLT processor is required. This software can be downloaded from <http://saxon.sourceforge.net/>.

Installation and configuration of Saxon, or any other external processor, will vary from product to product as well as between Windows and UNIX.



It is not necessary to use XSLT to create HTML applications with TierBroker. Skip to chapter 16 *Complex HTML Forms* to complete the basic version of this tutorial.

Saxon and XSLT are used to demonstrate how to link TierBroker to external applications to render HTML. Other rendering possibilities include using existing personalization or CRM applications to enrich either the XML document or the HTML page before returning it to the Browser.

15.2. HTML Input Form with XSLT Directive

```

<!-- (C) Universal Data Interface Corporation 2001-2002 -->
<HTML>
<HEAD>
  <TITLE>Stock Quote with XSLT</TITLE>
</HEAD>
<BODY>
  <FORM method='get' action='GetStockQuote$'>

  <!-- Format response with external processor -->
  <!-- Using OutputFileName$ - TB Generated
        RequestFileName$      - XML SOAP Request
        ResponseFileName$     - XML SOAP Response -->

  <INPUT
    type  ='hidden'
    name  ='udi:system'
    value ='saxon -o OutputFileName$ ResponseFileName$ StockQuote.xsl' />

  <H1><FONT color='red'>Get Stock Quote with XSLT</FONT></H1>

  <BR/>Symbol:
  <INPUT type='text' name='symbol' value='PEP' size='30' />

  <BR/>
  <INPUT type='submit' value="Submit" />
  </FORM>
</BODY>
</HTML>

```

The `udi:system` directive will be passed to the operating system shell. This directive uses the following macros that TierBroker expands when it generates the operating system command:

OutputFileName\$ TierBroker generated output file name. The external processor must create this file, which will be returned to the Browser or used in the next processing step as the *response*.

RequestFileName\$ The XML SOAP request document.

ResponseFileName\$ The XML SOAP response document.

Note that it is not necessary to use both the request and the response, and the example which is given uses only the SOAP response as input to XSLT.

☞ If both the request and the response are required in the finished document, they can still be rendered with XSLT by processing them in two steps. An alternative approach is to enrich the response with a scripting function before passing it to XSLT.

15.3. The XSLT Style Sheet

```
<xsl:stylesheet
  xmlns:n='urn:xmethods-delayed-quotes'
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">

  <xsl:output method="html"/>
  <xsl:variable name="title">Stock Quote</xsl:variable>

  <xsl:template match="/">
    <html>
      <head>
        <title><xsl:value-of select="$title"/></title>

        <style>
          <xsl:comment>

            /* default */ {
              font-family: arial,helvetica;
              color: black;
            }

            /* Add HTML Style Sheet Here */

          </xsl:comment>
        </style>

      </head>
      <body>
        <xsl:apply-templates/>
      </body>
    </html>
  </xsl:template>

  <xsl:template match="n:getQuoteResponse">
    <h2>Result: <xsl:value-of select="Result"/></h2>
  </xsl:template>

</xsl:stylesheet>
```

This is the simplest possible stylesheet, which renders the response as “Result: *Price*”.

15.4. To Run the XSLT Example

First run TierBroker from the operating system console:

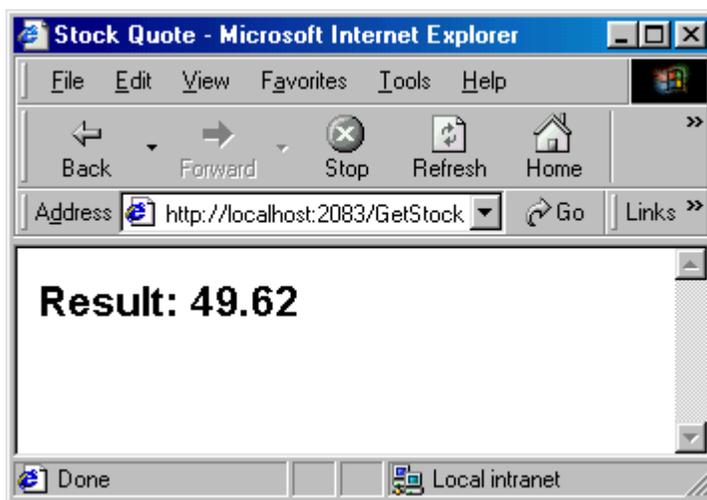
```
// Go to the tb\cbt\EasySoapNow directory
tb -UnStockQuote.udi -APlocalhost:2083 -ld3
```

Then access the input form with the Browser.

```
http://localhost:2083/StockQuote-XSL.html
```

Note that you must have an Internet connection to access the external stock quote price service.

15.5. The HTML Result



The HTML source:

```
<html xmlns:n="urn:xmethods-delayed-quotes">
  <head>
    <meta http-equiv="Content-Type" content="text/html; charset=utf-8">
    <title>Stock Quote</title>
    <style>
<!--
  /* default */ {
    font-family: arial,helvetica;
    color: black;
  }

  /* Add Additional HTML Style Sheet Information Here */
-->
    </style>
  </head>

  <body>
    <h2>Result: 49.62</h2>
  </body>
</html>
```

The XSLT example can be used as a template for creating complex stylesheets, including tables.

16. Using Scripted Functions to Render HTML

16.1. Using TBScript to Render HTML

This section continues the Stock Quote example application.



It is not necessary to use Scripted Functions to render HTML results. Skip to chapter 16 *Complex HTML Forms* to complete the basic version of this tutorial.

In this section a TBScript function is used to render the HTML page.

16.2. To Run the Scripting Example

First run TierBroker from the operating system console:

```
cd tb\cbt\EasySoapNow
tb -UnStockQuote.udi -APlocalhost:2083 -ld3
```

Then access the input form with the Browser.

```
http://localhost:2083/StockQuote-Fxn.html
```

Note that you must have an Internet connection to access the external stock quote price service.

16.3. HTML Input Form with Scripting Directive

```
<!-- (C) Universal Data Interface Corporation 2001-2002 -->
<HTML>
<HEAD>
  <TITLE>Stock Quote with TBScript Function</TITLE>
</HEAD>
<BODY>
  <FORM method='get' action='GetStockQuote$'>
    <!-- Format response with script function -->
    <INPUT
      type   ='hidden'
      name   ='udi:function'
      value  ='StockQuoteResponse' />
    <H1><FONT color='red'>Get Stock Quote with TBScript Function</FONT></H1>
    <BR/>Symbol:
    <INPUT type='text' name='symbol' value='PEP' size='30' />
    <BR/>
    <INPUT type='submit' value="Submit" />
  </FORM>
</BODY>
</HTML>
```

The `StockQuoteResponse()` function is defined in the TierBroker `StockQuote.ts` source file, which is a component in the `StockQuote.udi` project file.

16.4. The `StockQuoteResponse()` Function

```
VarChar StockQuoteResponse( TBNode soapRequest, TBNode soapResponse )
{
    var VarChar resp;

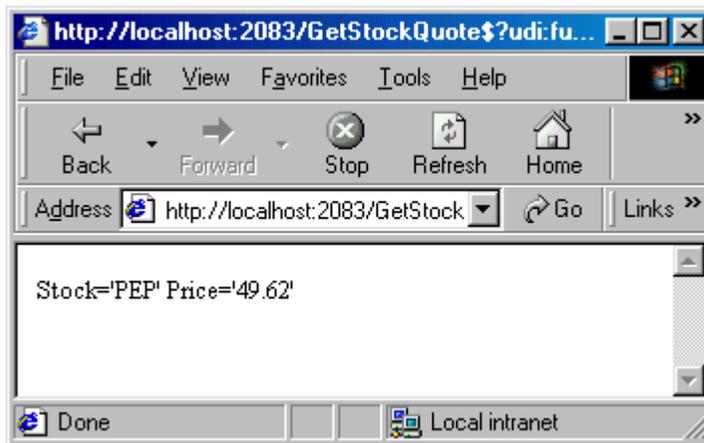
    // BreakPoint();
    // Comment BreakPoint() in to debug
    // SOAP method at web service request time...

    resp( "Stock='%s' Price='%s'",
        soapRequest.eval( '+symbol' ),
        soapResponse.eval( '+Result' ) );

    return resp;
}
```

This is a simple function that uses a C-Language style format control string to render the result.

16.5. The HTML Result



There are many more interesting applications for scripting, which can be used to enrich and validate information in the SOAP request and response.

17. Complex HTML Forms

17.1. AirFare Price Example

This example populates a complex SOAP request from a simple HTML input form. It renders the result into plain-text by combining information from both the request and the result.



Previous versions of this tutorial used a Web Service listed on XMethods.org in order to retrieve real-time travel quotes. The Web Service provider has stopped providing this service and this example has been recoded using direct access to www.Yahoo.com travel.

The AirFare Price example is broken down into two tutorial chapters. The first reviews this as an example of how to implement a complex input form. The next chapter reviews how to extract real-time information directly from an Internet web site.

17.2. Files Used in this Example

Primary Files

```
AirFare.ts           // TierBroker source code
AirFare.html        // HTML input form
AirFareResponse.html // TierBroker response template
```

These files represent the project source code.

Secondary Files

```
AirFare.htm         // Response page from Yahoo
AirFare.xml         // TierBroker XML SOAP response
```

These files represent the free-form HTML information that is extracted from the Yahoo web site and the result, which is the XML SOAP response.

17.3. To Run the AirFare Example

First run TierBroker from the operating system console:

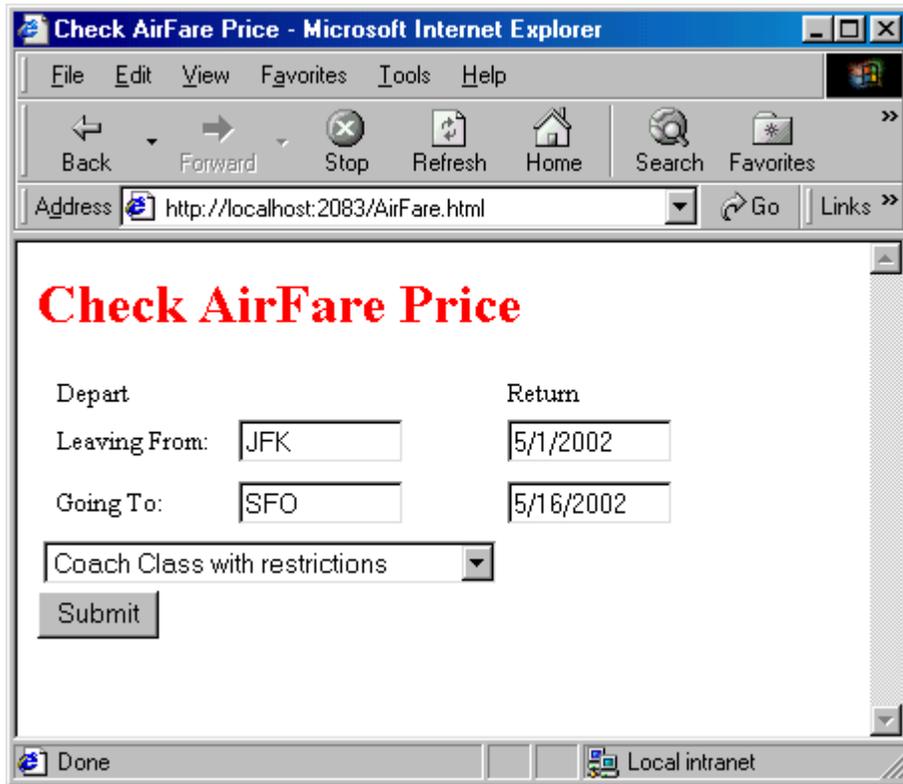
```
// Go to the tb\cvt\EasySoapNow directory
tb -UnAirFare.ts -Alocalhost:2083 -ld3
```

Then access the input form with the Browser.

```
http://localhost:2083/AirFare.html
```

Note that you must have an Internet connection to access the external AirFare price service.

17.4. The HTML Input Form



This form has the following HTML source code:

```

<!-- (C) Universal Data Interface Corporation 2001-2002 -->

<HTML>
<HEAD>
  <TITLE>Check AirFare Price</TITLE>
</HEAD>

<BODY>
  <H1><FONT color='red'>Check AirFare Price</FONT></H1>

  <FORM method='get' action='AirFare/getFlight'>

    <!-- Convert to plain text response -->
    <INPUT
      type = 'hidden'
      name = 'udi:template'
      value = 'AirFareResponse.html' />

    <TABLE border='0' cellPadding='2' cellSpacing='1'>
      <TR>
        <TD class='RH1'></TD>
        <TD class='RH2'>Depart</TD><TD/>
        <TD class='RH3'>Return</TD><TD/>
      </TR>

      <TR>
        <TD/><TD>Leaving From:</TD>
        <TD><INPUT type='text' name='fromAirport' value='JFK' size='10' /></TD>
        <TD><INPUT type='text' name='fromDate' value='5/1/2002' size='10' /></TD>
      </TR>

      <TR>
        <TD/><TD>Going To:</TD>
        <TD><INPUT type='text' name='toAirport' value='SFO' size='10' /></TD>
        <TD><INPUT type='text' name='toDate' value='5/16/2002' size='10' /></TD>
      </TR>

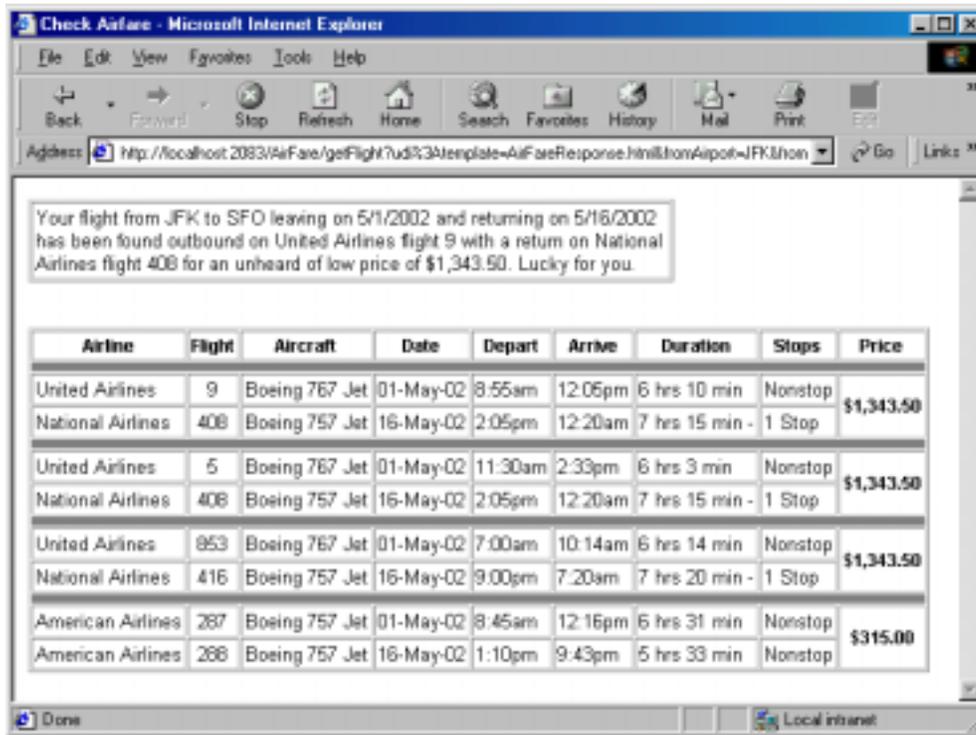
      <TR>
        <TD colspan='3'>
          <SELECT name="classSvc">
            <OPTION selected value="YR">Coach Class with restrictions</OPTION>
            <OPTION value="Y">Coach Class with no restrictions</OPTION>
            <OPTION value="CR">Business Class with restrictions</OPTION>
            <OPTION value="C">Business Class with no restrictions</OPTION>
            <OPTION value="FR">First Class with restrictions</OPTION>
            <OPTION value="F">First Class with no restrictions</OPTION>
          </SELECT>
        </TD>
      </TR>
    </TABLE>

    <INPUT type='submit' value="Submit" />
  </FORM>
</BODY>
</HTML>

```

In this case, a TierBroker proxy function is used as the SOAP Action to modify the information that is contained in the HTML form. A TierBroker template is used to render the response.

17.5. The AirFare HTML Response



Your flight from JFK to SFO leaving on 5/1/2002 and returning on 5/16/2002 has been found outbound on United Airlines flight 9 with a return on National Airlines flight 408 for an unheard of low price of \$1,343.50. Lucky for you.

Airline	Flight	Aircraft	Date	Depart	Arrive	Duration	Stops	Price
United Airlines	9	Boeing 767 Jet	01-May-02	8:55am	12:05pm	6 hrs 10 min	Nonstop	\$1,343.50
National Airlines	408	Boeing 757 Jet	16-May-02	2:05pm	12:20am	7 hrs 15 min -	1 Stop	
United Airlines	5	Boeing 767 Jet	01-May-02	11:30am	2:33pm	6 hrs 3 min	Nonstop	\$1,343.50
National Airlines	408	Boeing 757 Jet	16-May-02	2:05pm	12:20am	7 hrs 15 min -	1 Stop	
United Airlines	853	Boeing 767 Jet	01-May-02	7:00am	10:14am	6 hrs 14 min	Nonstop	\$1,343.50
National Airlines	416	Boeing 757 Jet	16-May-02	9:00pm	7:20am	7 hrs 20 min -	1 Stop	
American Airlines	287	Boeing 757 Jet	01-May-02	8:45am	12:16pm	6 hrs 31 min	Nonstop	\$315.00
American Airlines	286	Boeing 757 Jet	16-May-02	1:10pm	9:43pm	5 hrs 33 min	Nonstop	

The response is created from a TierBroker template.

17.6. The getFlight() XML SOAP Response

The HTML response from Yahoo is converted into the following XML SOAP response document.

```
<AirFareList>
  <airfare>
    <flight>
      <fare_code>WOE14NR8</fare_code>
      <ccy>USD</ccy>
      <price>336.50</price>
    </flight>
    <outbound>
      <airline>United Airlines</airline>
      <number>9</number>
      <eqp_name>Boeing 767 Jet</eqp_name>
      <dep_arp_code>JFK</dep_arp_code>
      <arr_arp_code>SFO</arr_arp_code>
      <dep_date>20020501</dep_date>
      <dep_time>8:55am</dep_time>
      <dep_day>Wed</dep_day>
      <arr_date>20020501</arr_date>
      <arr_time>12:05pm</arr_time>
      <stops>Nonstop</stops>
      <duration>6 hrs 10 min</duration>
    </outbound>
    <return>
      <airline>United Airlines</airline>
      <number>852</number>
      <eqp_name>Boeing 767 Jet</eqp_name>
      <dep_arp_code>SFO</dep_arp_code>
      <arr_arp_code>JFK</arr_arp_code>
      <dep_date>20020516</dep_date>
      <dep_time>11:25am</dep_time>
      <dep_day>Thu</dep_day>
      <arr_date>20020516</arr_date>
      <arr_time>7:51pm</arr_time>
      <stops>Nonstop</stops>
      <duration>5 hrs 26 min</duration>
    </return>
  </airfare>
  <-- More airfare elements omitted -->
</AirFareList>
```

The power of TierBroker is that it turns HTML into a usable XML document. Each airfare element contains information about a specific airfare.

17.7. The TierBroker AirFare Template

```

<!-- (C) Universal Data Interface Corporation 2002 -->
<HTML>
  <HEAD><!-- CSS Style Sheet Omitted --></HEAD>
  <BODY>
  <TABLE width='450' border='1' cellPadding='2' cellSpacing='1'>
  <TR><TD>
    <TEXT>Your flight from </TEXT>
    <TB request='+fromAirport' />
    <TEXT> to </TEXT>
    <TB request='+toAirport' />
    <TEXT> leaving on </TEXT>
    <TB request='+fromDate' />
    <TEXT> and returning on </TEXT>
    <TB request='+toDate' />
    <TEXT> has been found outbound on </TEXT>
    <TB response='+airfare[ 0 ]/outbound/airline' />
    <TEXT> flight </TEXT>
    <TB response='+airfare[ 0 ]/outbound[ 0 ]/number' />
    <TEXT> with a return on </TEXT>
    <TB response='+airfare[ 0 ]/return[ 0 ]/airline' />
    <TEXT> flight </TEXT>
    <TB response='+airfare[ 0 ]/return[ 0 ]/number' />
    <TEXT> for an unheard of low price of $</TEXT>
    <TB response='+airfare[ 0 ]/flight[ 0 ]/price' numberfmt="','.',',2' />
    <TEXT>. Lucky for you.</TEXT>
  </TD></TR>
</TABLE>

```

Notice that many rows select information from the first or *zeroth* airfare response, and that in cases where multiple outbound legs are present, only the first is used.

```

<TB response='+airfare[ 0 ]/outbound[ 0 ]/number' />

```

The + plus sign indicates that the SOAP response should be scanned for airfare elements. The airfare element does not need to appear under the root element, which is Envelope.

17.8. Flight Listings Table

HTML tables are covered in more detail in Chapter 19, *Rendering HTML Tables*. The table that is used in this template renders each `airfare` element into a row in the HTML table.

```

<-- AirFareResponse.html continued -->
<TABLE border='1' cellPadding='2' cellSpacing='1'>
  <TR>
    <TH>Airline</TH>
    <TH>Flight</TH>
    <TH>Aircraft</TH>
    <TH>Date</TH>
    <TH>Depart</TH>
    <TH>Arrive</TH>
    <TH>Duration</TH>
    <TH>Stops</TH>
    <TH>Price</TH>
  </TR>
  <TB response='+airfare'>
  <TR><TD colspan='9' class='SPC'><BR/></TD></TR>
  <TR>
    <TD><TB response='outbound/airline' /></TD>
    <TD class='CEN'><TB response='outbound/number' /></TD>
    <TD><TB response='outbound/eqp_name' /></TD>
    <TD><TB response='outbound/dep_date' datefmt ='YYYYMMDD;DDMMYY,-' /></TD>
    <TD><TB response='outbound/dep_time' /></TD>
    <TD><TB response='outbound/arr_time' /></TD>
    <TD><TB response='outbound/duration' /></TD>
    <TD><TB response='outbound/stops' /></TD>
    <TH valign='middle' rowspan='2'>
      $<TB response='flight/price' numberfmt="','.',2"/>
    </TH>
  </TR>
  <TR>
    <TD><TB response='return/airline' /></TD>
    <TD class='CEN'><TB response='return/number' /></TD>
    <TD><TB response='return/eqp_name' /></TD>
    <TD><TB response='return/dep_date' datefmt ='YYYYMMDD;DDMMYY,-' /></TD>
    <TD><TB response='return/dep_time' /></TD>
    <TD><TB response='return/arr_time' /></TD>
    <TD><TB response='return/duration' /></TD>
    <TD><TB response='return/stops' /></TD>
  </TR>
</TB>
</TABLE>
</BODY>
</HTML>

```

Note that this template does nothing more than build the plain text response from elements of the SOAP request and response.

17.9. Changing Date Formats

TierBroker allows dates to be parsed and reformatted.

```
<TB response='return/dep_date' datefmt ='YYYYMMDD;DDMMYY,-' /></TD>
```

The `datefmt` attributes gives the *old* date format and the *new* date format (separated by a semicolon) for the HTML template.

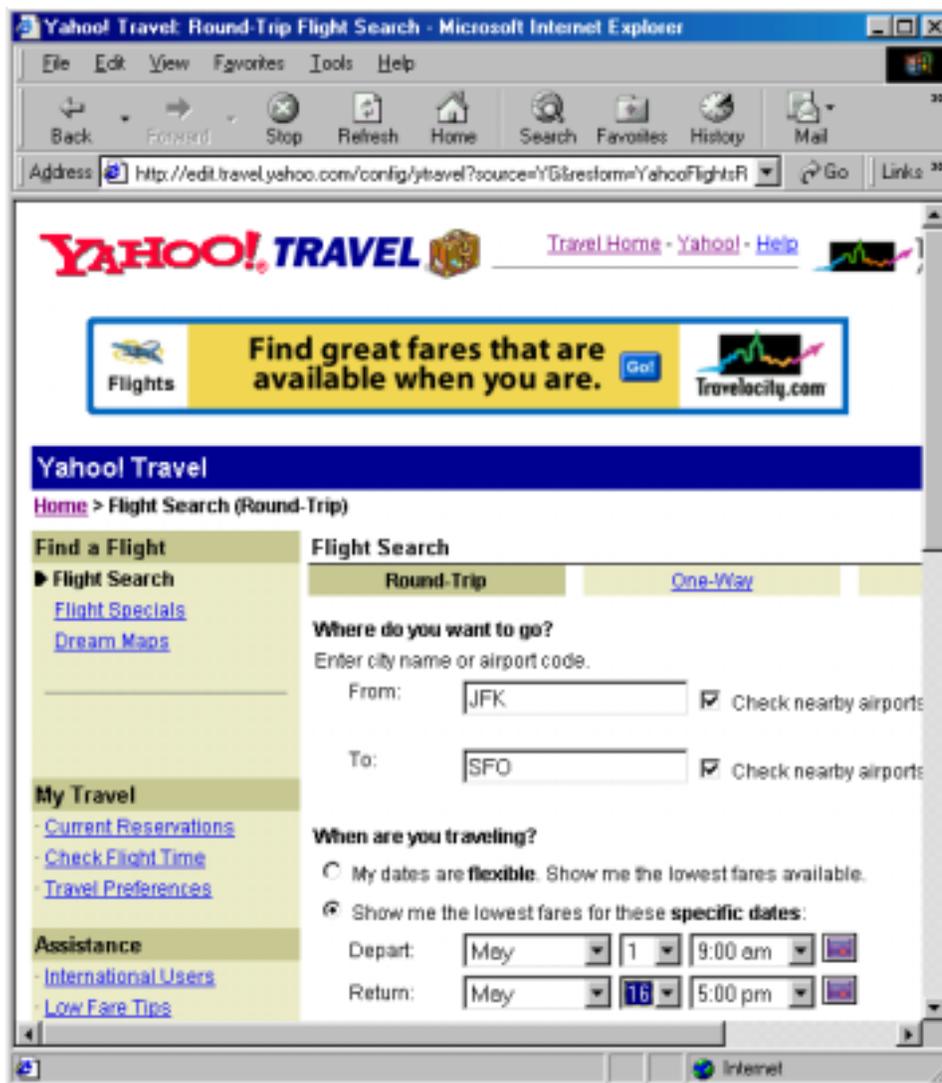
18. HTML Extraction and Screen Scraping

18.1. Turning Web Sites into Web Services

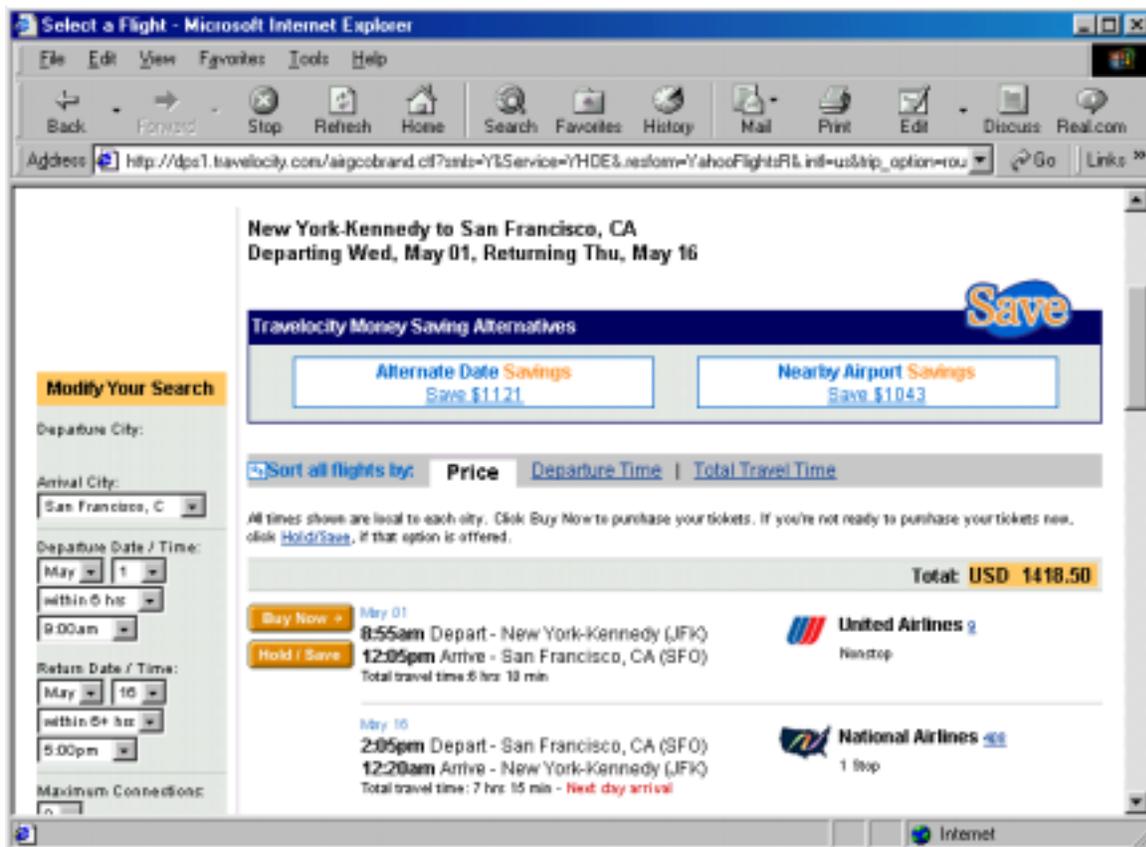
TierBroker can be used to turn any web site into a Web Service by accessing that web site and turning its HTML into useful clean XML.

☞ Pulling information from web sites is sometimes called *screen scraping*, because the information on the HTML page is loosely structured and must be scraped off the page.

The TierBroker approach to screen scraping is robust because it parses poorly formed HTML and turns it into XML, while preserving as much of the HTML markup as possible. TierBroker uses the same parsing rules as the Browser to create a canonical form of the HTML document. The first step is to execute a query on Yahoo (or any web site) to see how it formulates its GET or POST statement.



18.2. The Yahoo Travel Page



There are two important elements on this page. The GET which forms the query and the HTML page itself, which can be viewed and saved by right-clicking inside the page.

18.3. The HTTP GET Statement

At the top of the Browser, the GET statement appears as the URL after the word "Address". In the case where a POST is used, the POST format can be reconstructed by inspecting the HTML source code.

```
http://dps1.travelocity.com/aigcobrand.c?smIs=Y&Service=YHDE&.resform=YahooFlightsR
&.intl=us&trip_option=roundtrip&module=tripsrch&source=YG&adult_pax_cnt=1&chld_pax_cnt=
0&senior_pax_cnt=0&dep_arp_cd%281%29=JFK&arr_arp_cd%281%29=SFO&dep_arp_range%281%29=10
0&arr_arp_range%281%29=100&dep_dt_mn_1=May&dep_dt_dy_1=1&dep_tm_1=9%3a00am&dep_dt_mn_2
=May&dep_dt_dy_2=16&dep_tm_2=5%3a00pm&results=tripsrch&cls_svc=YR&pref_aln=all&aln_cd%
281%29=&aln_cd%282%29=&aln_cd%283%29=&num_cnx=0&tm_range=6
```

This may seem like one big mess. In reality, the query is a well formed markup, where each entry is name/value pair with the following syntax:

```
tag: (name=value&)*
```

In this case, the `tb/cbt/EasySoapNow/AirFare.ts` file begins with the following string:

```
// Travelocity HTTP GET/POST copied from browser.
// Note that query fields will be filled at runtime.

var VarChar TravelQuery =
    'http://dps1.travelocity.com/airgcobrand.ctl?smls=Y&'
    'Service=YHOE&'
    '.resform=YahooFlightsR&'
    '.intl=us&'
    'trip_option=roundtrip&'
    'module=tripsrch&'
    'source=YX&'
    'adult_pax_cnt=1&'
    'chld_pax_cnt=0&'
    'senior_pax_cnt=0&'
    'dep_arp_cd%%281%%29=%s& // From Airport
    'arr_arp_cd%%281%%29=%s& // To Airport
    'dep_dt_yr_1=%d& // From Date Year
    'dep_dt_mn_1=%s& // From Date Short Month (Jan)
    'dep_dt_dy_1=%d& // From Date Day
    'dep_tm_1=9%%3a00am&'
    'dep_dt_yr_2=%d& // To Date Year
    'dep_dt_mn_2=%s& // To Date Short Month (Jan)
    'dep_dt_dy_2=%d& // To Date Short Day
    'dep_tm_2=5%%3a00pm&'
    'cls_svc=%s& // Class of Service
    'pref_aln=all&'
    'aln_cd%%281%%29=&'
    'aln_cd%%282%%29=&'
    'aln_cd%%283%%29=&'
    'num_cnx=0&'
    'tm_range=+&'
    'SEQ=&'
    'dep_arp_range%%281%%29=150&'
    'arr_arp_range%%281%%29=150&';
```

This string will be used as the *format* of a `printf()` statement that will create the URL to retrieve the HTML page containing the flight information. The lines in **bold** indicate the places where information from the SOAP request will be used to populate the HTTP query.

18.4. The HTML Travel Query Response

```

<html>
<!-- airpdisp.pgd -->
<!-- BOX: 0016 -->
<!-- SL: -->
<head>
<!-- JavaScript and Style Sheet Omitted -->
</head>
<body bgcolor=#FFFFFF link=#000080>
<center>
<p>
<table border=0 cellpadding=2 cellspacing=0 width=100% name=toplinks>
<tbody>
<tr>

<td width="1%"><a href=http://travel.yahoo.com><img
rc=http://rg.travelocity.com.edgesuite.net/graphics/yho_bann.gif alt="Yahoo! Travel"
idth=285 height=30 border=0></a></td>
<td nowrap align=right width=100%><font face=arial,Helvetica size=-1><a
ref=http://travel.yahoo.com>Travel Home</a> - <a
ref=http://www.yahoo.com/>Yahoo!</a> - <a
ref=http://help.yahoo.com/help/trav>Help</a></font>
<hr size=1 noshade>
</td>
<td width=31 align=right><img
rc=http://rg.travelocity.com.edgesuite.net/graphics/yho_tcy3.gif
lt="Travelocity.com" width=162 height=31 border=0></td>
</tr>
</tbody>
</table name=toplinks>
<!--HDR BEG-->
<br>

REST OF DOCUMENT OMITTED...

```

The response page from Yahoo may appear difficult to follow.

 The easy way to inspect HTML pages is to first convert them into XHTML using TierBroker. Then they can be inspected using any XML editor.

TierBroker uses path notation – a superset of XPath – to extract information from the HTML response.

18.5. The XML SOAP Response

This is the XML document created by TierBroker from the HTML input document using the `getFlight()` SOAP function.

```
<AirFareList>
  <airfare>
    <flight>
      <fare_code>WOE14NR8</fare_code>
      <ccy>USD</ccy>
      <price>336.50</price>
    </flight>
    <outbound>
      <airline>United Airlines</airline>
      <number>9</number>
      <eqp_name>Boeing 767 Jet</eqp_name>
      <dep_arp_code>JFK</dep_arp_code>
      <arr_arp_code>SFO</arr_arp_code>
      <dep_date>20020501</dep_date>
      <dep_time>8:55am</dep_time>
      <dep_day>Wed</dep_day>
      <arr_date>20020501</arr_date>
      <arr_time>12:05pm</arr_time>
      <stops>Nonstop</stops>
      <duration>6 hrs 10 min</duration>
    </outbound>
    <return>
      <airline>United Airlines</airline>
      <number>852</number>
      <eqp_name>Boeing 767 Jet</eqp_name>
      <dep_arp_code>SFO</dep_arp_code>
      <arr_arp_code>JFK</arr_arp_code>
      <dep_date>20020516</dep_date>
      <dep_time>11:25am</dep_time>
      <dep_day>Thu</dep_day>
      <arr_date>20020516</arr_date>
      <arr_time>7:51pm</arr_time>
      <stops>Nonstop</stops>
      <duration>5 hrs 26 min</duration>
    </return>
  </airfare>
  <!-- Other airfare elements omitted -->
</AirFareList>
```

The `AirFareList` is returned by the SOAP function, which is a proxy for the travel service.

18.6. Wrapping Legacy Systems as Web Services

☞ Wrapping the Yahoo Travel Service as a Web Service using TierBroker demonstrates the principles of wrapping any legacy system using any transport protocol.

In this case HTTP and HTML are used as the source of the legacy data. However, the principles of wrapping and extraction could equally well be applied to any type of legacy transaction.

18.7. The *getFlight()* SOAP Function

The following source code:

```
C:\Program Files\UDICo\TierBroker\Tb\cbt\EasySoapNow\AirFare.ts
```

contains the source code for the SOAP service.

```
_operation TBNode getFlight( VarChar fromAirport,
                             Date      fromDate,
                             VarChar toAirport,
                             Date      toDate,
                             VarChar classSvc = 'YR' )
{
    var VarChar query;

    // Create the query string
    query( TravelQuery, fromAirport, toAirport,
          fromDate.getYear(), fromDate.getShortMonth(), fromDate.getDay(),
          toDate.getYear(), toDate.getShortMonth(), toDate.getDay(),
          classSvc );
```

This code *printf's* the values from the SOAP service into the query string. At this point, the value of query could be pasted into the URL of the Browser to bring back the travel page result.

```
var VarChar r( getFile( query ) ); // Pull from the web...
var TBNode n( r );                // Convert from HTML to XML
```

The `getFile()` function can be used with a URL as well as a file name. This one statement pulls back the HTTP response, which is an HTML document, and saves it in the string variable `r`. The next line takes that string and inserts it as markup into a *Document Object Model* (DOM) by using `TBNode` class instance.

```
// Useful debugging info...
// r.save( 'AirFare.htm' ); // Save the original HTML
// n.save( 'AirFare.xml' ); // Save as useful XML/XHTML
// n.save( 'AirFare2.htm' ); // Save as well-formed HTML
```

These steps are commented out, but indicate how intermediate states can be saved to facilitate the development and debugging process.

```
// Flight instructions...
var TBPPath inst( n, "+table/tr[ 2 ]/td"
                 "/input[@name['fare_bss_cd','cur_cd','total_price']]"
                 "/@value" );
```

A `TBPPath` class is used to retrieve data from the HTML response. This path literally means:

Find any table where the third row has a TD where the INPUT has a name attribute with either 'fare_bss_cd', 'cur_cd', or 'total_price' and return the contents of the value attribute.

Note that these values are returned in exactly the order in which they appear on the HTML page.

```
// Other info, duration and number of stops...
var TBPath info( n, "=table[@name['flight_info*']]/tr/td/span/a/@href" );
var TBPath time( n, "=table[@name['flight_info*']]/tr/td[@*align['top']]/b" );
var TBPath stop( n, "=table[@name['flight_info*']]/tr/td[@class['content-sm']]" );
```

Other paths are used to extract various elements from the table called `flight_info`.

```
var TBNode outb( 'outbound' );
var TBNode retn( 'return' );
var TBNode response( 'AirFareList' );
```

Documents which are used to create each `airfare` element in the `AirFareList`.

```
while (inst.next()) {
```

For every set of flight instructions...

```
info.next();
outb.putGetPost( info.getNode() );
```

Take the next `info` item, which is an `href`, and insert that GET/POST query text into a node.

```
//syslog << info.getNode().value() << cr;
```

In a debugging scenario, the value of that node can be printed to the console screen.

```
info.next();
retn.putGetPost( info.getNode() );
//syslog << info.getNode().value() << cr;
```

Do the same for the flight return.

```
var TBNode airfare( 'airfare' );
```

Create a new `airfare` element.

```
airfare.insert( 'flight/fare_code', inst.getNode().value() );
airfare.insert( 'flight/ccy', inst.next().value() );
airfare.insert( 'flight/price', inst.next().value() );
```

Insert values from the flight instructions.

```
airfare.insert( 'outbound/airline', outb( 'aln_name' ) );
airfare.insert( 'outbound/number', outb( 'flt_num' ) );
airfare.insert( 'outbound/eqp_name', outb( 'eqp_name' ) );
airfare.insert( 'outbound/dep_arp_code', outb( 'dep_arp_code' ) );
airfare.insert( 'outbound/arr_arp_code', outb( 'arr_arp_code' ) );
airfare.insert( 'outbound/dep_date', outb( 'dep_dt' ) );
airfare.insert( 'outbound/dep_time', time.next().value() );
airfare.insert( 'outbound/dep_day', outb( 'rqs_dow' ) );
airfare.insert( 'outbound/arr_date', outb( 'dep_dt' ) );
airfare.insert( 'outbound/arr_time', time.next().value() );
airfare.insert( 'outbound/stops', stop.next().value() );
airfare.insert( 'outbound/duration', extract( stop.next().value(), ':*' ) );
```

Insert values from the outbound information, time and number of stops. The `extract()` function is used to pull just the flight time from the flight duration statement.

```

airfare.insert( 'return/airline',      retn( 'aln_name' ) );
airfare.insert( 'return/number',      retn( 'flt_num' ) );
airfare.insert( 'return/eqp_name',    retn( 'eqp_name' ) );
airfare.insert( 'return/dep_arp_code', retn( 'dep_arp_code' ) );
airfare.insert( 'return/arr_arp_code', retn( 'arr_arp_code' ) );
airfare.insert( 'return/dep_date',    retn( 'dep_dt' ) );
airfare.insert( 'return/dep_time',    time.next().value() );
airfare.insert( 'return/dep_day',     retn( 'rqs_dow' ) );
airfare.insert( 'return/arr_date',    retn( 'dep_dt' ) );
airfare.insert( 'return/arr_time',    time.next().value() );
airfare.insert( 'return/stops',       stop.next().value() );
airfare.insert( 'return/duration',    extract( stop.next().value(), ':*' ) );

```

Create the return values.

```

        response.insert( airfare );
    }

```

Insert the `airfare` element into the `AirFareList` response, and then loop for every set of flight instructions offered for this query.

```

        // HTML web page has been
        // Converted to useful XML

        // For debugging...
        // syslog << response;
        // response.save( 'AirFare.xml' );

        return response;
    }
    /***** end of AirFare.ts *****/

```

Return the response document as the Body of the SOAP envelope.

19. Rendering HTML Tables

19.1. RTStockQuote Price Example

This example populates a SOAP request from a simple HTML input form. It renders the result into a table stock price entries by combining information from both the request and the result.

19.2. Files Used in this Example

```
RTStockQuote.ts           // TierBroker source code
RTStockQuote.html        // HTML input form
RTStockQuoteResponse.html // TierBroker response template
```

19.3. To Run the RTStockQuote Example

First run TierBroker from the operating system console:

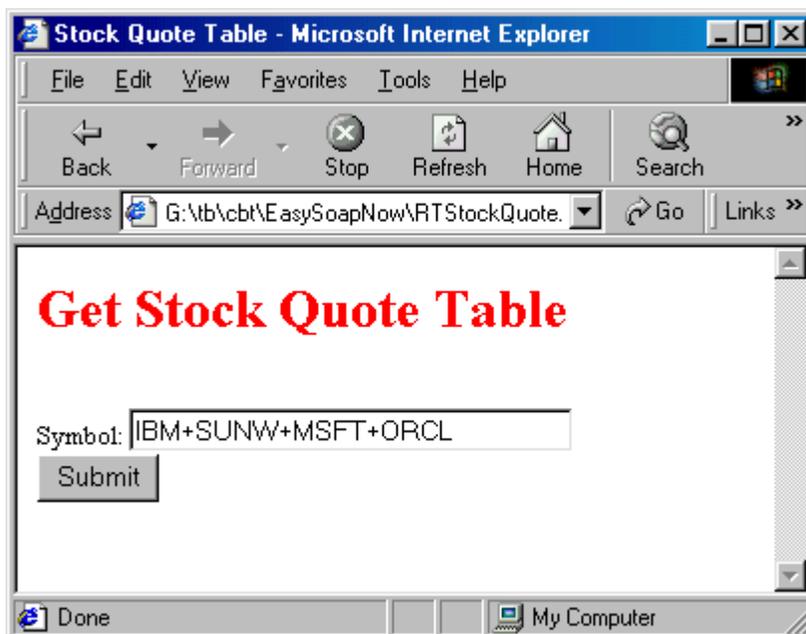
```
// Go to the tb\cbt\EasySoapNow directory
tb -UnRTStockQuote.ts -APllocalhost:2083 -ld3
```

Then access the input form with the Browser.

```
#http://localhost:2083/RTStockQuote.html
```

Note that you must have an Internet connection to access the external stock quote price service.

19.4. The HTML Input Form



This form has the following HTML source code:

```
<!-- (C) Universal Data Interface Corporation 2001-2002 -->
<HTML>
<HEAD>
  <TITLE>Stock Quote Table</TITLE>
</HEAD>
<BODY>
  <FORM method='post' action='RTStockQuote/getStockList'>

  <!-- Adjust some request/response values -->
  <INPUT
    type = 'hidden'
    name = 'udi:function'
    value = 'RTStockQuoteFixValues' />

  <!-- Select Response Format Template -->
  <INPUT
    type = 'hidden'
    name = 'udi:template'
    value = 'RTStockQuoteResponse.html' />

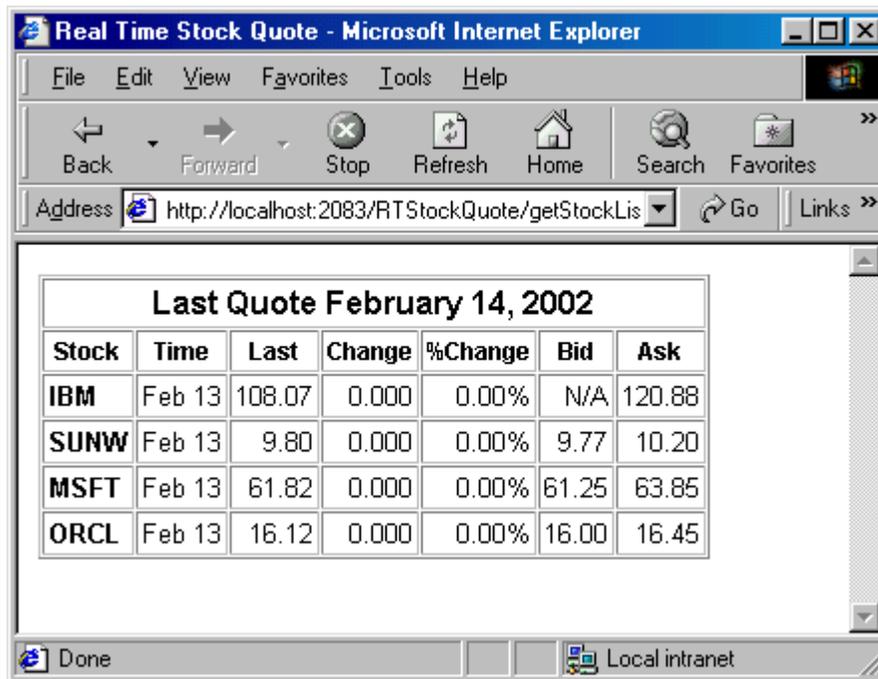
  <H1><FONT color='red'>Get Stock Quote Table</FONT></H1>

  <BR/>Symbol:
  <INPUT type='text' name='symbolList' value='IBM+SUNW+MSFT+ORCL' size='30' />

  <BR/>
  <INPUT type='submit' value="Submit" />
  </FORM>
</BODY>
</HTML>
```

This example uses two steps to render the HTML response. First, a TierBroker function is used to modify the information that is contained in the SOAP response. Then, a TierBroker template is used to render the response into HTML.

19.5. The RTStockQuote HTML Response



The screenshot shows a Microsoft Internet Explorer window titled "Real Time Stock Quote - Microsoft Internet Explorer". The address bar displays "http://localhost:2083/RTStockQuote/getStockLis". The main content area contains a table titled "Last Quote February 14, 2002". The table has seven columns: Stock, Time, Last, Change, %Change, Bid, and Ask. The data rows are for IBM, SUNW, MSFT, and ORCL.

Stock	Time	Last	Change	%Change	Bid	Ask
IBM	Feb 13	108.07	0.000	0.00%	N/A	120.88
SUNW	Feb 13	9.80	0.000	0.00%	9.77	10.20
MSFT	Feb 13	61.82	0.000	0.00%	61.25	63.85
ORCL	Feb 13	16.12	0.000	0.00%	16.00	16.45

The response is created from a TierBroker template.

19.6. The TierBroker RTStockQuote Template

```

<!-- (C) Universal Data Interface Corporation 2002 -->
<HTML>
  <HEAD><!-- CSS Style Sheet Omitted --></HEAD>

  <BODY>
  <TABLE border='1' cellPadding='2' cellSpacing='1'>

    <TR><TH class='OVR' colspan='7'>
      <TEXT>Last Quote </TEXT><TB response='+quotedate' />
    </TH></TR>
    <TR>
      <TH>Stock</TH><TH>Time</TH><TH>Last</TH><TH>Change</TH>
      <TH>%Change</TH><TH>Bid</TH><TH>Ask</TH>
    </TR>

    <!-- The '+' means all stock_quote elements in this response.
         The template under TB is applied to each instance of stock_quote.
         A template within a template could also be nested. -->

    <TB response='+stock_quote'>
      <TR>
        <TD class='SYM'>
          <TB response='symbol' />
          <TEXT udi:sample='1'>IBM</TEXT>
        </TD>
        <TD>
          <TB response='time' />
          <TEXT udi:sample='1'>11:14AM</TEXT>
        </TD>
        <TD>
          <TB response='last' />
          <TEXT udi:sample='1'>90.09</TEXT>
        </TD>
        <TD>
          <TB response='change' />
          <TEXT udi:sample='1'>0.14</TEXT>
        </TD>
        <TD>
          <TB response='pctchange' />
          <TEXT udi:sample='1'>+0.12%</TEXT>
        </TD>
        <TD>
          <TB response='bid' />
          <TEXT udi:sample='1'>90.08</TEXT>
        </TD>
        <TD>
          <TB response='ask' />
          <TEXT udi:sample='1'>90.12</TEXT>
        </TD>
      </TR>
    </TB>
  </TABLE>
</BODY>
</HTML>

```

Note that this template builds each row in the finished table. Also note that templates can be applied recursively, creating tables within tables. Also note that this template uses *sample text* to make the job of building the template easier if HTML designer tools, such as *DreamWeaver* are used.

20. Currency Trader Application

20.1. *Bringing it All Together*

The Currency Trader application uses many of the principles developed in the preceding chapters, in order to create a complete application that demonstrates some advanced principles of Web Site construction using TierBroker and Web services.

20.2. *Accessing the Currency Trader Application from the Internet*

Readers who have not downloaded TierBroker can still access the Currency Trader Application.

<http://www.udico.com:2083/CcyLogin.html>

The Currency Trader Application is available on the UDICo Web Site using TierBroker as the host.

20.3. *Primary Files Used in this Example*

```
CcyTrader.udi           // TierBroker project file 66 lines
CcyTrader.ts           // TBScript source code 199 lines
CcyLogin.html          // User login HTML page 32 lines
CcyMain.html           // Trading screen HTML page 99 lines
CcyMainResponse.html  // TierBroker response template 76 lines
```

Small is Beautiful

The entire application – including comments, version documentation and HTML cascading style sheets – is only 265 lines of TierBroker and 216 lines of plain-text HTML. An equivalent application developed with either ASP/.NET or JSP/J2EE tools would require more than 5-10 times the amount of deliverable work product in compiled source code and HTML.

20.4. *Support Files Used in this Example*

```
CcyPwdGen.ts           // Utility to obtain password keys
CcyAcctList.csv        // List of default example accounts
CcyTrader.csv          // Default application data
CcyTrader.mdb          // Application database
```

Note that the customer accounts use a secure identifier that is a unique hash code of the user name and password. This customer id acts as a one way key – it can be derived from the user name and password, but they cannot be reconstructed using the key. This technique means that it is not necessary to store customer passwords in the application database.

Creating Customer Accounts

The customer keys can be generated using the following script:

```
// Go to the tb/cbt/EasySoapNow directory
tb -UnCcyPwdGen.ts -AwCcyAcctList.csv
```

Creating the CcyTrader.mdb File

The application database can be created by running the following workflow:

```
// Go to the tb/cbt/EasySoapNow directory
tb -UnCcyTrader.udi -AnCreateTables
```

This workflow populates the sample database using the information in CcyTrader.csv.

20.5. Running the CcyTrader Application

From the command prompt enter:

```
// Go to the tb/cbt/EasySoapNow directory
tb -UnCcyTrader.udi -APlocalhost:2083 -ld3
```

This command loads the TierBroker Server with the Currency Trader application. The TierBroker Server console should display:

```
TierBroker V4.21
(C) 1993-2002 Universal Data Interface Inc.
Loading Project:Version/Workflow<CurrencyTrader:/>
Evaluation version licensed for non-commercial use.
Expires on 04/01/2002

tb>
```

The Currency Trader application is ready to be accessed from the Browser.

20.6. Accessing Currency Trader from the Browser

The following URL is the Currency Trader login page.

```
http://localhost:2083/CcyLogin.html
```

The Browser should display:



The username/password kyuyama/kyuyama is a valid account in the sample database. Other valid accounts include simon/simon and shirley/shirley.

20.7. Secure Login Using a One-Way Key

Invalid username/password combinations will be rejected by the TierBroker Server. However, the password is not stored in the application database.

☞ Do not store production passwords on disk! Make sure that they are never logged to a console as plain-text!

TierBroker does not persist passwords – it does not even store its own passwords in its memory core at runtime. Database passwords, which it must preserve in its runtime image, are encrypted.

☞ The TierBroker Server cannot be hacked for passwords, even by analyzing the runtime image of the memory core.

The Currency Trader application demonstrates key principles in the design of secure systems. By itself, it is not bullet proof, because the dialog between the Browser and the server is not secure. However, by

adding secure sockets, or by using the TierBroker Compression and Encryption Proxy⁶, a bullet proof application can be constructed.

20.8. Currency Trader Main Screen

Click the Submit button to proceed to the CcyMain.html trading screen.

日本語版ティアブローカーは、
2002年6月に発売されます。

**Currency
Trader** irashimase **Yuyama-sama.**
Honjitsu wa UDICo Currency Japan Ltd.
he yokoso okoshi kudasaimashite
makotoni arigato gozaimasu.

February 3, 2002 zukeno kawase reto
wa tsugino hyo no tori desu.

Currency	FX Rate	Description
AUD	1.9143	AUSTRALIAN DOLLARS
DEM	2.7852	DEUTSCHE MARKS
GBP	0.6891	BRITISH POUNDS
FRF	7.5830	FRENCH FRANCS
JPY	128.2100	JAPANESE YEN
ITL	2,198.0600	ITALIAN LIRA
USD	1.0000	US DOLLARS

Quantity: from: to:

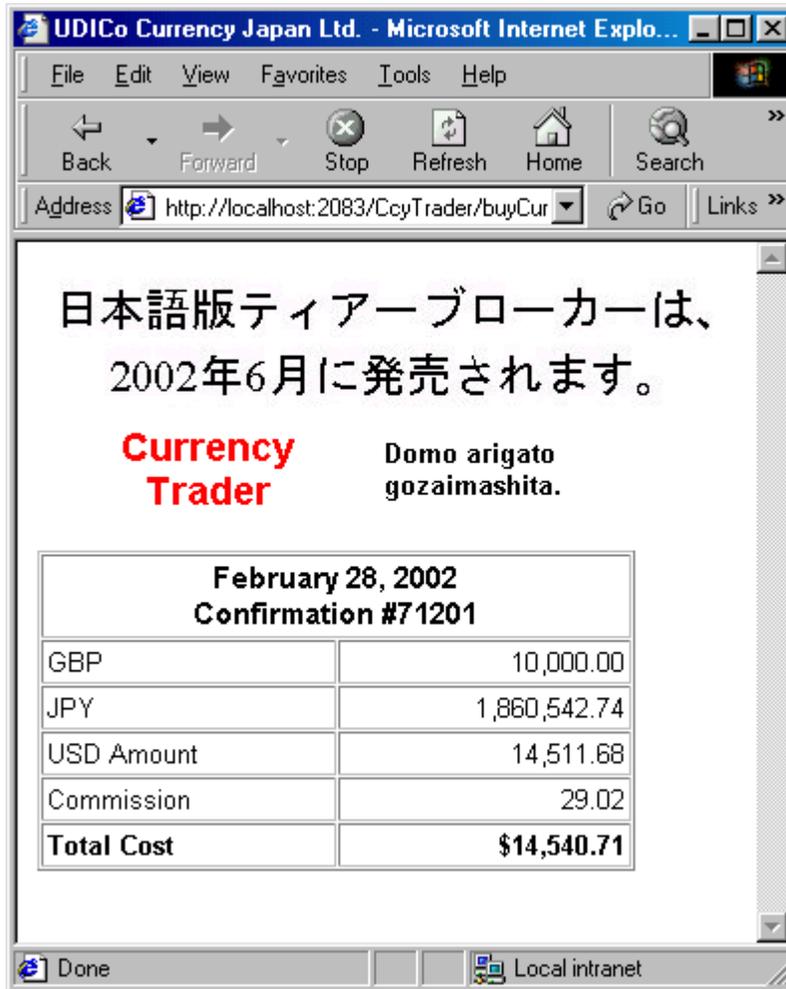
Done Local intranet

This application allows the trader to convert units of currency.

⁶ The TierBroker encryption and compression proxy is scheduled for general availability in Q3/2002.

20.9. Execute a Trade

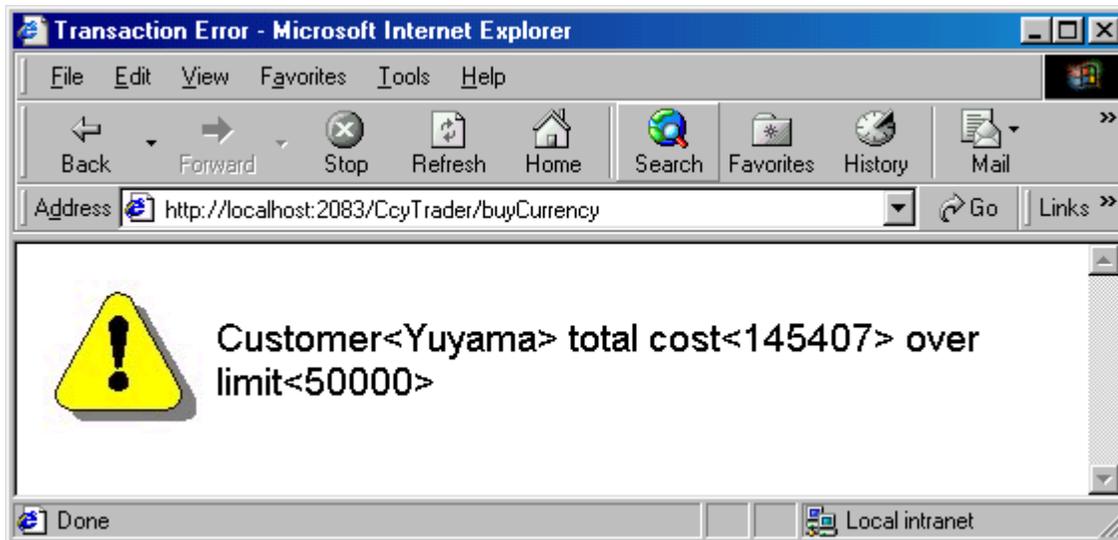
Enter 10000 for Quantity and convert from GBP to JPY. Then click the EXECUTE button.



The Confirmation Screen displays the result, including the transaction confirmation number.

20.10. Exception Handling at the Application Level

Use the Browser Back button to return to the trading screen. Change the Quantity to 100000 GBP and press the EXECUTE button again. This time the Browser will display:



The customer has gone over-limit and the trade has been rejected.

20.11. Source Code Review for Currency Trader

The following sections reviews points of interest in the Currency Trader source code.



It is not necessary to understand the Currency Trader Application in detail to complete the basic version of this tutorial. Running the examples given above completes the basic material.

Building applications as a developer does require an understanding of the material that follows. Readers who wish to develop even moderately complex Forms, Tables, Frames or Web Sites should complete this section.

20.12. The CcyLogin.html Screen

The following is the HTML source code for the Currency Trader Application login screen.

```

<!-- (C) Universal Data Interface Corporation 2001-2002 -->
<HTML>
<HEAD>
  <TITLE>Currency Trader</TITLE>
</HEAD>
<!-- The user login is handled by the SOAP function CcyTraderLogin(),
      and the SOAP response is rendered into the CcyMain.html page. -->
<BODY>
  <FORM method='post' action='CcyTrader/CustomerLogin'>
    <H1><FONT color=red>Currency Trader Login</FONT></H1>
    <!-- Render Main Page -->
    <INPUT
      type  ='hidden'
      name  ='udi:template'
      value ='CcyMain.html' />
    <BR>User ID:
    <INPUT type=text name=userid value='kyuyama' size= 30>
    <BR>Password:
    <INPUT type=password name=password value='kyuyama' size= 30>
    <BR>
    <INPUT type=submit value="Submit" />
  </FORM>
</BODY>
</HTML>

```

This form uses a basic TierBroker pattern:

1. The ACTION of the POST is a SOAP function.
2. The HTML response is rendered using a TierBroker Template.
3. The Form INPUT controls are named after the SOAP function parameters.

In this case, TierBroker is hosting the CcyTrader Web Service, and the CustomerLogin function operation of the SOAP port is being accessed.

 Note that default values are supplied for userid and password for demonstration purposes only!

The response is rendered with the CcyMain.html template. Rendering directives are placed in hidden controls and must be a parameter of the form ACTION. A TierBroker directive is indicated by the udi namespace prefix, and the supported directives include udi:template, udi:system and udi:function, all of which are demonstrated in the *Easy Soap Now!* tutorial.

20.13. The CustomerLogin() SOAP Function

The source code for the CustomerLogin() function is defined in CcyTrader.ts.

```

_operation _volatile TBNode CustomerLogin( VarChar userid, VarChar password )
{
    var TBNode      response( 'Result' );
    var Customer    customer( userid, password );
    var UserSession session ( customer.custId );

    response.insert( 'SessionId', session.sessionId );
    response.insert( customer.getXml( 'title,lastName' ) );
    response.insert( 'FXRatelist' );

    for (var TBObjItr i( 'FXRate', 'LOGIN( TrxnDB )' ); i.next(); )
        response.insertByPath( 'FXRateList', *i );

    return response;
}

```

The `_operation` keyword indicates that this is a SOAP operation, and that this function should be published as a part of the SOAP definitions/service. The `_volatile` keyword indicates that this function should always be executed, and that the results can never be cached.

DOM and XPATH in TBScript

The TBNode datatype is equivalent to an XML document or DOM. The response document is constructed in the body of the CustomerLogin() function. The TBNode and TBPPath Computer Based Training modules review DOM and XPATH in TierBroker.

```

var TBNode      response( 'Result' );

```

This line creates an XML document instance with a local variable name of `result` and an XML document root node with an element name of `Result`.

Classes in TBScript

TierBroker supports classes with multiple inheritance, *isa* and *hasa* relationships, as well as many other features of object oriented programming. The following classes:

```

Class Customer {
    VarChar(13)  custId;
    VarChar(32)  userid;
    Date         asOfDate;
    Char(8)      title;
    VarChar(64)  firstName;
    VarChar(64)  lastName;
    Number       limit;

    fxn void Customer( VarChar userid, VarChar password );
};
Class(1) UserSession {
    VarChar(32)  custId;
    Integer      sessionId;
    Datetime     lastTime;

    fxn void UserSession( VarChar custId );
};

```

Are used in the `CustomerLogin()` function.

```
var Customer    customer( userid, password );
var UserSession session ( customer.custId );
```

In this case, the `Customer` constructor is used to *lookup* the customer data record and validate the `userid` and `password`. The `UserSession` class is used to establish a session record. Details on these two classes follow later in this chapter.

20.14. Populating the XML Document

```
response.insert( 'SessionId', session.sessionId );
response.insert( customer.getXml( 'title,lastName' ) );
```

These two statements fix the `SessionId` and `Customer` elements in the document. At this point, the `Result` document contains the following elements.

```
<Result>
  <SessionId>71601</SessionId>
  <Customer>
    <title>Mr.</title>
    <lastName>Yuyama</lastName>
  </Customer>
```

Notice how the complex `Customer` element is created from:

```
customer.getXml( 'title,lastName' )
```

TierBroker classes are persistent in XML and can be rendered using a simple list of class data members.

Retrieving Lists for SQL or Documents

```
response.insert( 'FXRatelist' );

for (var TBObjItr i( 'FXRate', 'LOGIN( TrxnDB )' ); i.next(); )
  response.insertByPath( 'FXRateList', *i );
```

This code creates the `FXRateList` element, and then populates it with the Foreign eXchange rates in the `TrxnDB` transaction database. The *signature* of this database is defined in the `CcyTrader.udi` TierBroker project file. The `TBObjItr` class is an iterator that can process SQL logins, with a `where` clause, as well as XML documents and CSV files from either the file system or a URI.

```
<FXRatelist>
  <FXRate>
    <ccyId>1001</ccyId>
    <asOfDate>02/03/2002</asOfDate>
    <currencyCode>AUD</currencyCode>
    <exchangeRate>1.9143</exchangeRate>
    <description>AUSTRALIAN DOLLARS</description>
  </FXRate>
  <!-- and so on... -->
</FXRateList>
```

This is an example from the `FXRateList` in the SOAP response.

20.15. The Complete CustomerLogin() SOAP Response

The complete SOAP response has the following content:

```
<?xml version='1.0' encoding='UTF-8'?>
<SOAP-ENV:Envelope xmlns:SOAP-ENV='http://schemas.xmlsoap.org/soap/envelope/'>
  <SOAP-ENV:Body>
    <CustomerLoginResponse xmlns="http://localhost:2083/CcyTrader">
      <Result>
        <SessionId>71601</SessionId>
        <Customer>
          <title>Mr.</title>
          <lastName>Yuyama</lastName>
        </Customer>
        <FXRatelist>
          <FXRate>
            <ccyId>1001</ccyId>
            <asOfDate>02/03/2002</asOfDate>
            <currencyCode>AUD</currencyCode>
            <exchangeRate>1.9143</exchangeRate>
            <description>AUSTRALIAN DOLLARS</description>
          </FXRate>
          <FXRate>
            <ccyId>1002</ccyId>
            <asOfDate>02/03/2002</asOfDate>
            <currencyCode>DEM</currencyCode>
            <exchangeRate>2.7852</exchangeRate>
            <description>DEUTSCHE MARKS</description>
          </FXRate>
          <!-- and so on... -->
        </FXRatelist>
      </Result>
    </CustomerLoginResponse>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

This response is generated by TierBroker after:

```
return response;
```

the return statement. Note that the response can be a TBNODE, a text file or a string containing XML. If a class object is returned, then it is converted to XML and wrapped in the SOAP response envelope.

20.16. The Customer Class Detail

Customer Class Declaration

```

Class Customer {
    VarChar(13)  custId;
    VarChar(32)  userid;
    Date         asOfDate;
    Char(8)      title;
    VarChar(64)  firstName;
    VarChar(64)  lastName;
    Number       limit;

    fxn void Customer( VarChar userid, VarChar password );
} ;

```

Note that TBScript data types are at a high level compared to the fundamental data types of C++ or Java. The Date and Datetime classes have sophisticated date math, which is documented in the *TierBroker Reference Manual* and on-line TierBroker Server HTML help pages.

Customer Class Constructor

```

Class UserLogin {
    VarChar userid;
    VarChar password;
} ;

void Customer::Customer( VarChar userid, VarChar password )
{
    var UserLogin user;
    user.userid = userid;
    user.password = password;

    custId = user.getHashCode();

    // Validate customer in db and lookup record
    // If not found return SOAP fault as response

    if (!fromLookup( 'CustomerLkup' ))
        throw( 'User<%s> not found', userid );
}

```

The constructor function takes the `userid` and `password` and populates the complete customer record from the transaction database using a Lookup definition, which is defined in the `CcyTrader.udl` project file.

```

    custId = user.getHashCode();

```

This statement constructs a unique one-way key from the `UserLogin` class instance. This is the unique customer identifier and is used as the *lookup key* to retrieve the customer record.

```

    if (!fromLookup( 'CustomerLkup' ))
        throw( 'User<%s> not found', userid );

```

If the customer record is not found an exception is returned⁷. In this case, the error is formatted using the default SOAP fault template.

```
tb/bin/cmd/SoapFault.html
```

Is the default SOAP Fault response template.

20.17. Exception Handling in TBScript

The `throw()` statement is used to throw exceptions. These errors are recorded in the TierView⁸ transaction logging database.

The Default SOAP Fault Template

```
tb/bin/cmd/SoapFault.html
```

The default template can be overridden by any template with the same name as long as it occurs *earlier* in the project file include path.

```
<!-- (C) Universal Data Interface Corporation 2002 -->
<HTML>
  <HEAD>
    <TITLE>Transaction Error</TITLE>
    <STYLE>
      BODY { FONT: 10pt verdana }
      SELECT { FONT: 10pt verdana }
      TD { FONT: 10pt arial }
      INPUT { FONT: 10pt verdana }
      TH { COLOR: black; FONT: bold 20pt verdana }
      .MSG { COLOR: black; TEXT-ALIGN: left; FONT: 14pt arial bold; }
      .PIC { TEXT-ALIGN: right; WIDTH: 50; }
    </STYLE>
  </HEAD>

  <BODY>
    <TABLE border='0' cellPadding='2' cellSpacing='1' width='500'>
      <TR>
        <TD class='PIC'><IMG border='0' src='SoapFault.jpg' /></TD>
        <TD class='MSG'><TB response='+faultstring' /></TD>
      </TR>
    </TABLE>
  </BODY>
</HTML>
```

⁷ Formatted strings using C-Language style `printf()` codes are useful, but optional part of the TBScript language. Unlike C programs, TierBroker formatted strings are type-checked by the server at runtime and cannot cause a program fault.

⁸ TierView is the transaction status monitor built into the TierBroker Server. This component is reviewed in a separate CBT module.

SOAP Fault Example

An example SOAP Fault:

```
<?xml version='1.0' encoding='UTF-8'?>
<SOAP-ENV:Envelope xmlns:SOAP-ENV='http://schemas.xmlsoap.org/soap/envelope/'>
  <SOAP-ENV:Body>
    <SOAP-ENV:Fault>
      <faultcode>SOAP-ENV:Server</faultcode>
      <faultstring>Processing Error</faultstring>
      <faultactor>TierBroker V4.21</faultactor>
      <detail>File<bbadinoff> Ln<bbadinoff> Col<bbadinoff>
User<bbadinoff> not found</detail>
    </SOAP-ENV:Fault>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

This template renders a login error as:



User<bbadinoff> not found

In the case where a valid customer login is used, then the customer record is returned.

20.18. Creating the Session Handle

```
Class(1) UserSession {
  VarChar(32) custId;
  Integer    sessionId;
  Datetime   lastTime;

  fxn void UserSession( VarChar custId );
} ;

// Queue is defined in CcyTrader.udi
var TBQueue sessionOutputQueue( 'SessionOutputQueue' );

// UserSession class constructor
void UserSession::UserSession( VarChar custId )
{
  sessionId = seqIdNext( 'TrxnDB:SessionId' );
  this.custId = custId;
  lastTime   = getSysDatetime();

  // Weak session affinity
  sessionOutputQueue << this;
}
```

Notice that the session handle is created when the UserSession instance is *put* to the sessionOutputQueue. The queue definition is reviewed later in this tutorial.

20.19. Management of Stateless Sessions

The technique of creating a session state record and caching it in the production database means that logical end user sessions are stateless with respect to the TierBroker Server.

☞ TierBroker could be used in a server farm, because session state information is not cached in the Server memory.

The queue definition for the session handle ensures that only one session per user may be active at time, although this is an application level decision that can be easily modified if desired.

20.20. The buyCurrency() SOAP Function

The buyCurrency() function is used to complete a currency transaction after the user has successfully logged in, and established an open session.

```

_operation _volatile Transaction buyCurrency( Integer sessionId,
                                             Number quantity,
                                             VarChar fromCcy,
                                             VarChar toCcy )
{
    var Transaction txn;
    var SessionData data( sessionId );

    // These are a lookup, but could also be a SOAP call
    var Number fromRate = lookup( 'FXRateLkup', fromCcy );
    var Number toRate   = lookup( 'FXRateLkup', toCcy );

    // Calculate exact amounts...
    txn.trxnId       = seqIdNext( 'TrxnDB:TrxnId' );
    txn.custId       = data.customer.custId;
    txn.inputDate    = getSysDatetime();
    txn.baseCcyQty   = quantity / fromRate;
    txn.commission   = txn.baseCcyQty * 0.002;
    txn.totalCost    = txn.baseCcyQty + txn.commission;

    txn.fromCcy      = fromCcy;
    txn.fromCcyQty   = quantity;

    txn.toCcy        = toCcy;
    txn.toCcyQty     = toRate * txn.baseCcyQty;

    // Round after the fact...
    txn.toCcyQty     = fround( txn.toCcyQty, 2 );
    txn.baseCcyQty   = fround( txn.baseCcyQty, 2 );
    txn.commission   = fround( txn.commission, 2 );
    txn.totalCost    = fround( txn.totalCost, 2 );

    if (txn.totalCost > data.customer.limit)
        throw( 'Customer<%s> total cost<%.01f> over limit<%.01f>',
              data.customer.lastName, txn.totalCost, data.customer.limit );

    // Complete transaction
    txnOutputQueue << txn;

    return txn;
}

```

This function builds upon the previous material. Notice that the transaction is checked against the customer trading limit. This could easily be extended to check account balance or use a SOAP function to perform credit scoring or make a trade credit decision.

20.21. The CcyTrader.udl Project File

The CcyTrader.udl TierBroker project file contains the data definitions and workflows that are applied in the TBScript SOAP functions.

```

<Project id = 'CurrencyTrader'
  author    = '(C) Universal Data Interface Corporation 2002'>

<Script>
#include 'CcyTrader.ts'
</Script>

<!-- SQL Signature Definitions -->
<SqlLoginList>
  <SqlLogin id    = 'TrxnDB'
    sqlPlatform  = 'ODBC'
    dbName       = 'CcyTrader.mdb' />
</SqlLoginList>

<!-- Input and output queue definitions -->
<QueueList>
  <SqlQueue id    = 'SessionOutputQueue'
    queueType     = 'output'
    useDBName     = 'TrxnDB'
    deleteFirstFlg = 'True'
    arraySize     = '1' />

  <SqlQueue id    = 'TrxnOutputQueue'
    queueType     = 'output'
    useDBName     = 'TrxnDB'
    arraySize     = '1' />
</QueueList>

<!-- Lookups to external databases and files -->
<LookupList>
  <Lookup id      = 'UserSessionLkup'
    lookupClass   = 'UserSession'
    lookupIndex   = 'sessionId'
    cacheType     = 'LoadAndDiscard'
    whereClause   = "LOGIN( TrxnDB )" />

  <Lookup id      = 'CustomerLkup'
    lookupClass   = 'Customer'
    lookupIndex   = 'custId'
    whereClause   = "LOGIN( TrxnDB )" />

  <Lookup id      = 'FXRateLkup'
    lookupClass   = 'FXRate'
    lookupIndex   = 'currencyCode'
    lookupReturn  = 'exchangeRate'
    whereClause   = "LOGIN( TrxnDB )" />
</LookupList>

```

CcyTrader.udi Contined...

```

<!-- Workflows used to process transaction batches -->
<WorkflowList>
<Workflow id='CreateTables'>
  <doc>This workflow creates and loads the MS Access database.</doc>

  <OnLoad>
    <Sql sqlName      = 'TrxnDB'
        createTables = 'Customer,UserSession,FXRate,Transaction' />
  </OnLoad>

  <InputList>
    <CsvQueue clsFilterList = 'Customer' fileName = 'CcyTrader.csv' />
    <CsvQueue clsFilterList = 'FXRate'   fileName = 'CcyTrader.csv' />
  </InputList>

  <OutputList>
    <SqlQueue
      useDBName = 'TrxnDB' />
  </OutputList>
</Workflow>
</WorkflowList>
</Project>

```

The following sections review this project in detail.

20.22. TierBroker Project Structure

```

<Project id = 'CurrencyTrader'
  author   = '(C) Universal Data Interface Corporation 2002'>

<Script>
#include 'CcyTrader.ts'
</Script>

```

The `Project` element begins the project. The `Script` element includes the `CcyTrader.ts` file that defines the business logic used in this application.

☞ The relationship between the project and the script is the same as the relationship between HTML and JavaScript.

The TierBroker project is used to define *application structure*, while the script file defines *application behavior*. The project also contains data definitions for items such as SQL logins, lookups and queues.

20.23. SQL Login Definitions

```

<!-- SQL Signature Definitions -->
<SqlLoginList>
  <SqlLogin id      = 'TrxnDB'
            sqlPlatform = 'ODBC'
            dbName    = 'CcyTrader.mdb' />
</SqlLoginList>

```

All of the TierBroker Project Classes are referenced in the on-line server documentation. This tutorial only reviews the basic parameters of these elements. In this case, the *signature* defined by the `SqlLogin:id` of `TrxnDB` refers to an MS Access database file.

20.24. Queue Definitions

Input and output queues are used to create a layer of abstraction between the TBScript functions and the details of how objects are read or written to databases, files, sockets and messaging middleware.

```
<!-- Input and output queue definitions -->
<QueueList>
  <SqlQueue id      = 'SessionOutputQueue'
    queueType      = 'output'
    useDBName      = 'TrxnDB'
    deleteFirstFlg = 'True'
    arraySize      = '1' />

  <SqlQueue id      = 'TrxnOutputQueue'
    queueType      = 'output'
    useDBName      = 'TrxnDB'
    arraySize      = '1' />
</QueueList>
```

In this case, both output queues will flush their contents whenever they are written to. This behavior is established by the `SqlQueue::arraySize`. In the case of the `SessionOutputQueue`, the previous user session will be purged with a SQL delete statement. This behavior is established by `SqlQueue::deleteFirstFlg`.

20.25. Lookups to External SQL Databases and Files

Lookups are one of the most common application activities. They all have the common form of taking a single or multi-column key and returning as single or multiple values from a *dictionary*. TierBroker lookups can access database, files and URIs. Caching parameters are used to indicate of lookup results can be cached in memory.

```
<!-- Lookups to external databases and files -->
<LookupList>
  <Lookup id      = 'UserSessionLkup'
    lookupClass  = 'UserSession'
    lookupIndex  = 'sessionId'
    cacheType    = 'LoadAndDiscard'
    whereClause  = "LOGIN( TrxnDB )" />

  <Lookup id      = 'CustomerLkup'
    lookupClass  = 'Customer'
    lookupIndex  = 'custId'
    whereClause  = "LOGIN( TrxnDB )" />

  <Lookup id      = 'FXRateLkup'
    lookupClass  = 'FXRate'
    lookupIndex  = 'currencyCode'
    lookupReturn = 'exchangeRate'
    whereClause  = "LOGIN( TrxnDB )" />
</LookupList>
```

Notice that in the case of the `FXRateLkup` the currency code is used to return the exchange rate value.

```
var Number fromRate = lookup( 'FXRateLkup', fromCcy );
```

This statement sets the floating point number `fromRate` equal to the result of that lookup.

20.26. Project Workflows

Project workflows are not a direct component of the Currency Trader Application. Instead, they are used to build the sample MS Access database from the raw data in the `CcyTrader.csv` spreadsheet file.

```
<!-- Workflows used to process transaction batches -->
<WorkflowList>
<Workflow id='CreateTables'>
  <doc>This workflow creates and loads the MS Access database.</doc>

  <OnLoad>
    <Sql sqlName      = 'TrxnDB'
        createTables = 'Customer,UserSession,FXRate,Transaction' />
  </OnLoad>

  <InputList>
    <CsvQueue clsFilterList = 'Customer' fileName = 'CcyTrader.csv' />
    <CsvQueue clsFilterList = 'FXRate'   fileName = 'CcyTrader.csv' />
  </InputList>

  <OutputList>
    <SqlQueue
      useDBName = 'TrxnDB' />
  </OutputList>
</Workflow>
</WorkflowList>
```

The single `Project::Workflow, CreateTables` can be executed with:

```
cd tb/cbt/EasySoapNow           // CD to the EasySoapNow tutorial directory
tb -UnCcyTrader.udl -AnCreateTables // Execute the CreateTables Workflow
```

A detailed discussion of workflow is beyond the scope of this tutorial, but is covered in detail in other TierBroker CBT modules.

20.27. Currency Trader Conclusions

This completes the *Easy Soap Now!* white paper and tutorial. The Appendix contains information about the TierBroker Integrated Development Environment or IDE.

21. Appendix I: The TierBroker IDE

21.1. *Product Overview*

The TierBroker IDE is the developers' workbench that accompanies the TierBroker server, used in the previous examples. The aim of the IDE, like any development tool, is to provide a working environment that improves the productivity of developers as they create, test and deploy applications. In the world of TierBroker, of course, there is no compiled code, so the 'source code' that developers create is actually a combination of XML and TB Script documents. TierBroker IDE includes:

- A full featured text editor with programmable macros
- Color coding for TierBroker documents
- Context completion for XML documents
- Graphical wizards for automatically generating XML blocks
- Visual previews of transformation rules and process workflow
- A TB Script visual debugger

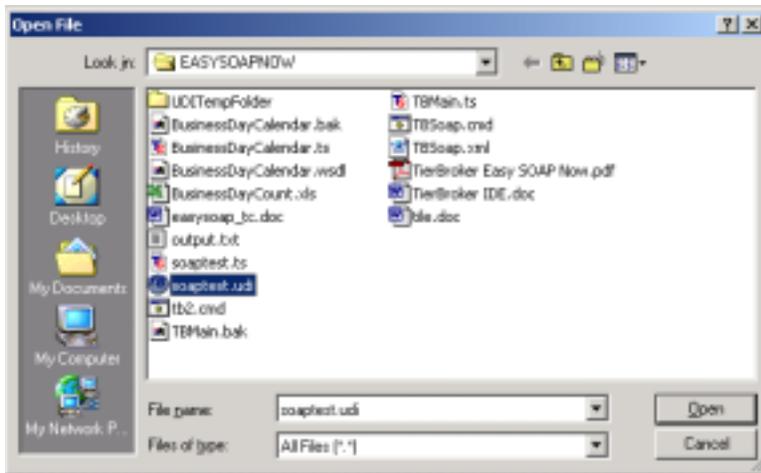
These XML and TBScript files could be created in any editor, but the TierBroker IDE is tailored for the development of these source files and offers countless features that make TierBroker developers more productive.

21.2. *Stepping Through Your Script*

Also included in the installation of TierBroker is an additional example that illustrates how the IDE's debugger can be used to step through and debug your SOAP services. This example illustrates how to build a test harness for debugging a block of scripting and introduces the use of XML as a structured method of describing TierBroker applications.

21.3. *Opening the Sample Project*

Start the IDE and select *File*, Open



Select the file titled *soaptest.udi*. This file is a complete TierBroker project file, which references the function contained in *BusinessDayCalendar.ts* script file

21.4. Understanding the TierBroker Project

While TB Script files can be loaded into the server and executed, an alternative approach is to structure applications as projects. A project document is simply an XML document adheres to the TierBroker document structure.

Project files may contain numerous elements that are not relevant to this example. Please refer to the on-line TierBroker help for additional information. For the purposes of this example we have selected a minimum set of elements to illustrate the example.

```
<!-- SOAP Test Project -->
<!-- (C) Universal Data Interface Corporation 2001 -->
<Project prjName = 'SOAP Test Project'>
<Doc>This project is designed to demonstrate debugging SOAP services.</Doc>
```

This initial set of elements simply at the top of the project document simply document the purpose of the project. The *Project* element is a required item, as is the attribute *prjName*. This serves as a heading under which all other elements will fall.

The *Doc* element is used to include comments, and may be used anywhere in the project document.

```
<Script>
    #include 'BusinessDayCalendar.ts'
</Script>
```

The *Script* element indicates that a block of TB Script will follow. In this case we use an include statement to reference the `BusinessDayCalendar.ts` file that contains the SOAP service scripting.

```

<Workflow id='SOAPTTest'>
  <InputList>
    <NullQueue/>
  </InputList>
  <OnLoad>
    <Script><![CDATA[
      void onOccur( )
      {
        BreakPoint();
        BusinessDayCount( '1/1/2002', '1/31/2002' );
      }
    ]]></Script>
  </OnLoad>
</Workflow>

```

The *Workflow* element defines a process that will be executed. Typically this would include a list of input, outputs and logic that is to be applied to transform data. In this case, our workflow is greatly simplified.

The *InputList*, ordinarily directs the server where it can find input data. In this case we specify only a *NullQueue* indicating that no source data is present.

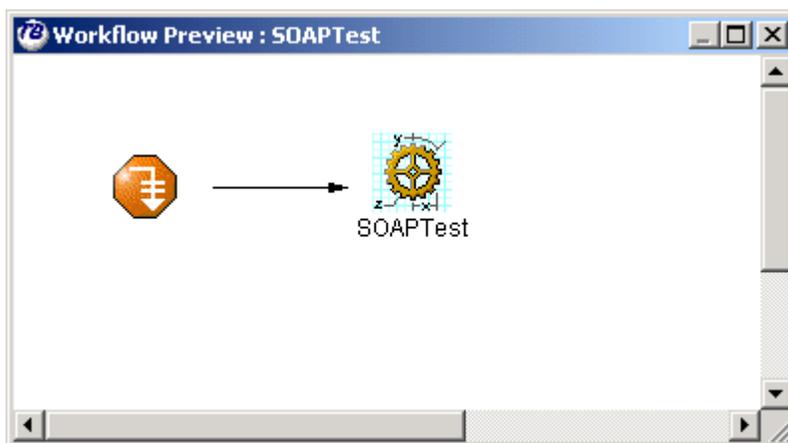
The *OnLoad* element represents an event that is triggered when our workflow begins. In this case we specify that at the beginning of this workflow, we should invoke the TBScript function called *onOccur*.

The *onOccur* function contains only two statements. The first indicate a debugging breakpoint. The second is a call to the `BusinessDayCount` function with the two literal dates passed in as arguments.

21.5. Project Summary

In effect what we defined is a process that has no input or output, but will invoke the function that in previous examples acted as our SOAP service. While this may seem senseless, this provides a framework for testing TBScript functions in a batch mode, rather than running them as real-time services.

To see a pictorial representation of this simple process. Place the mouse over the Workflow titled `SOAPTTest` in the project tree, on the left hand side of the IDE. Right click and select *Workflow Preview...*



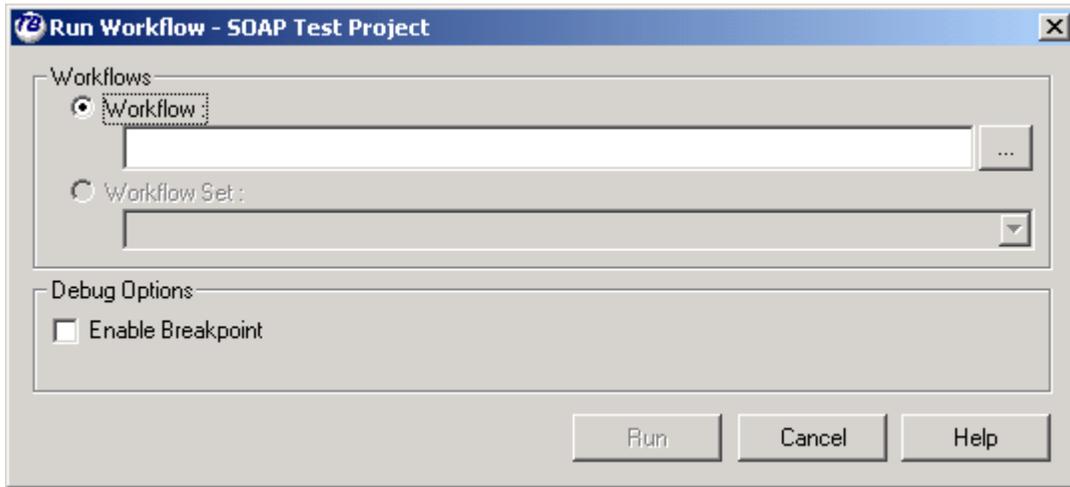
The Preview capability is useful for documentation as well as providing a visual reference. This particular project is not especially interesting, but in more complex workflows, the preview is invaluable.

21.6. Running the Workflow

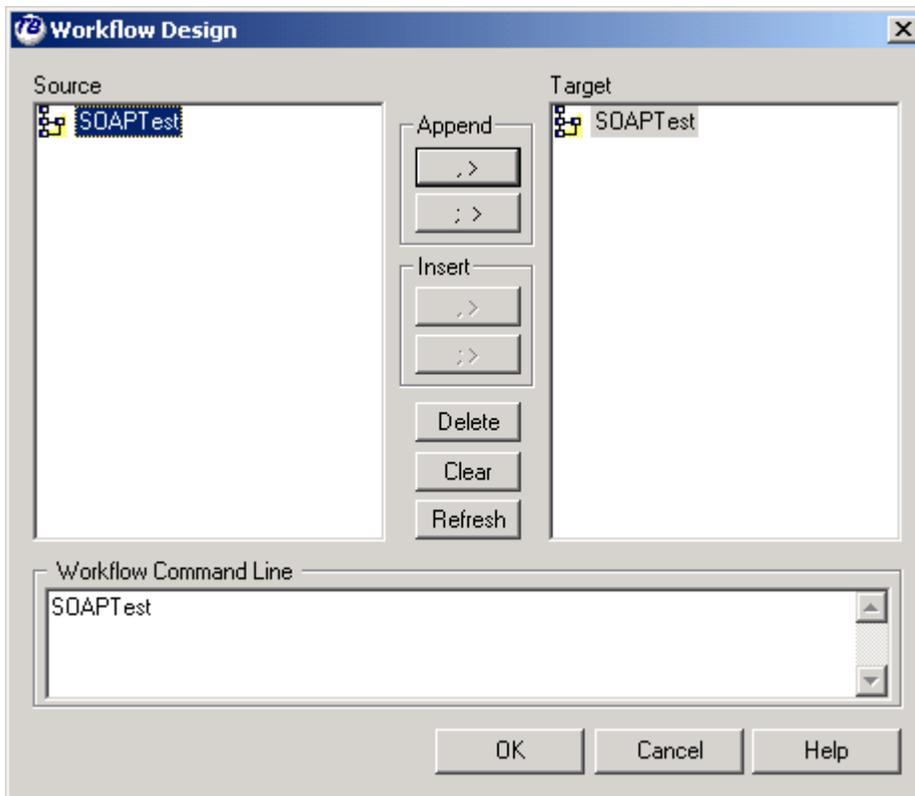
To run the workflow click on the toolbar button that looks like...

<insert button image here>

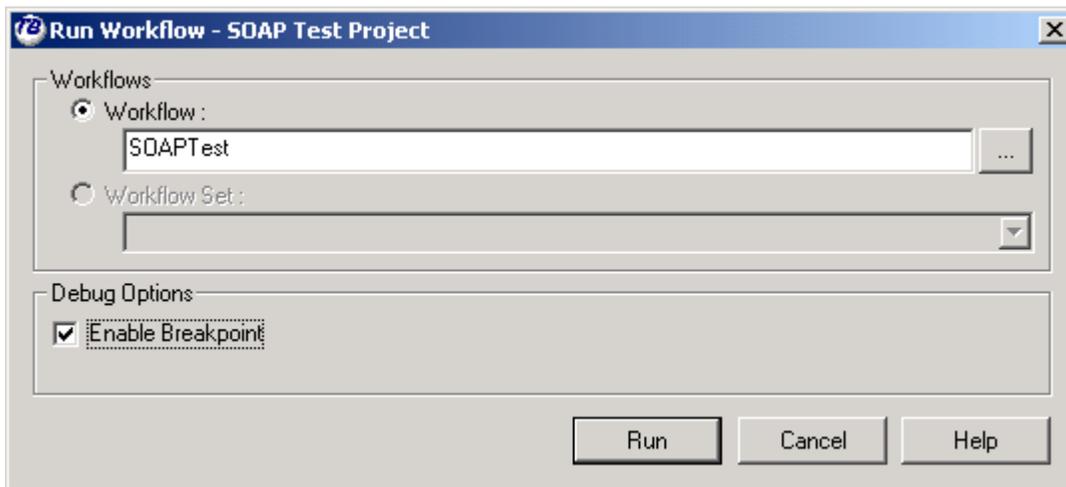
In the following dialog box click browse to select a workflow.



Select the only available workflow, which is *SOAPTest*. Click OK.



Ensure that the EnableBreakpoint option is selected. This will cause the server to break at the specified statements in the TBScript blocks.

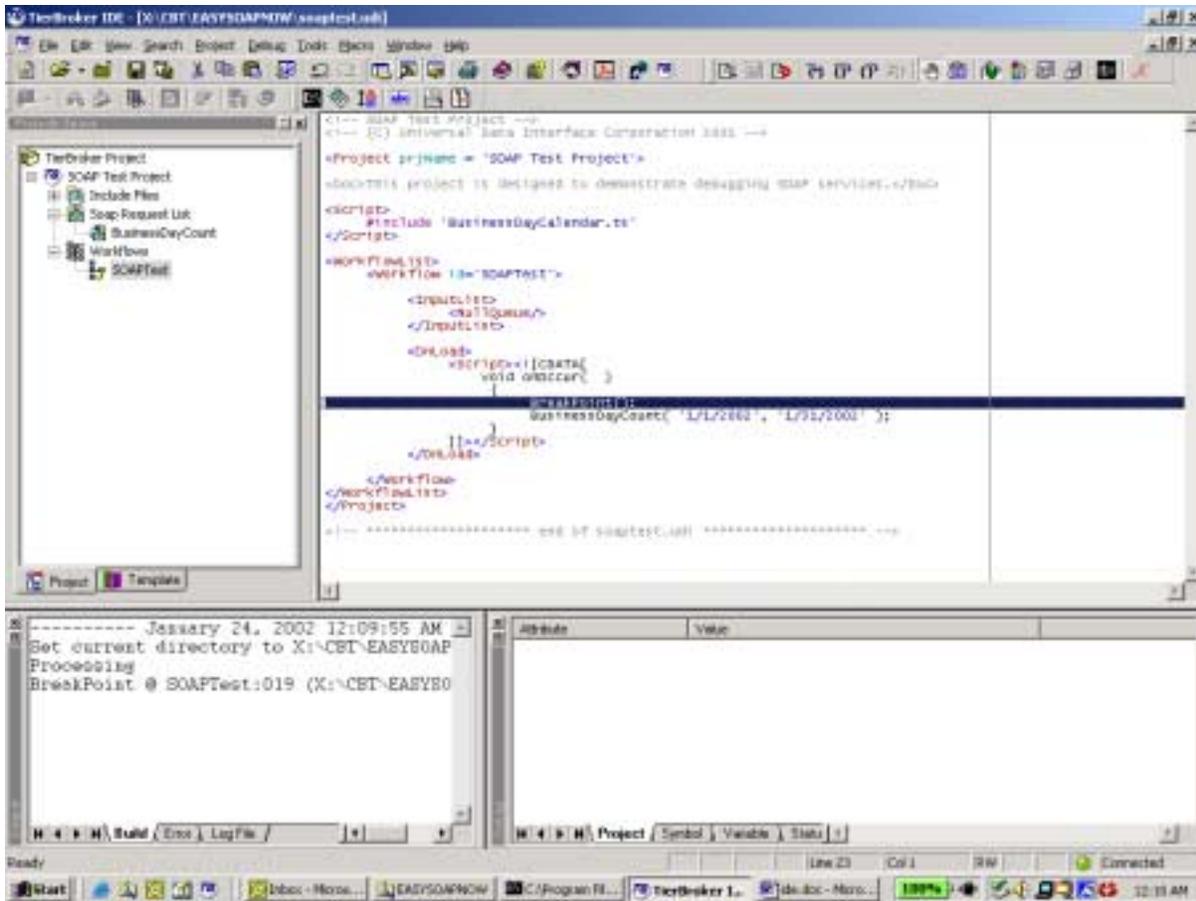


Click Run.

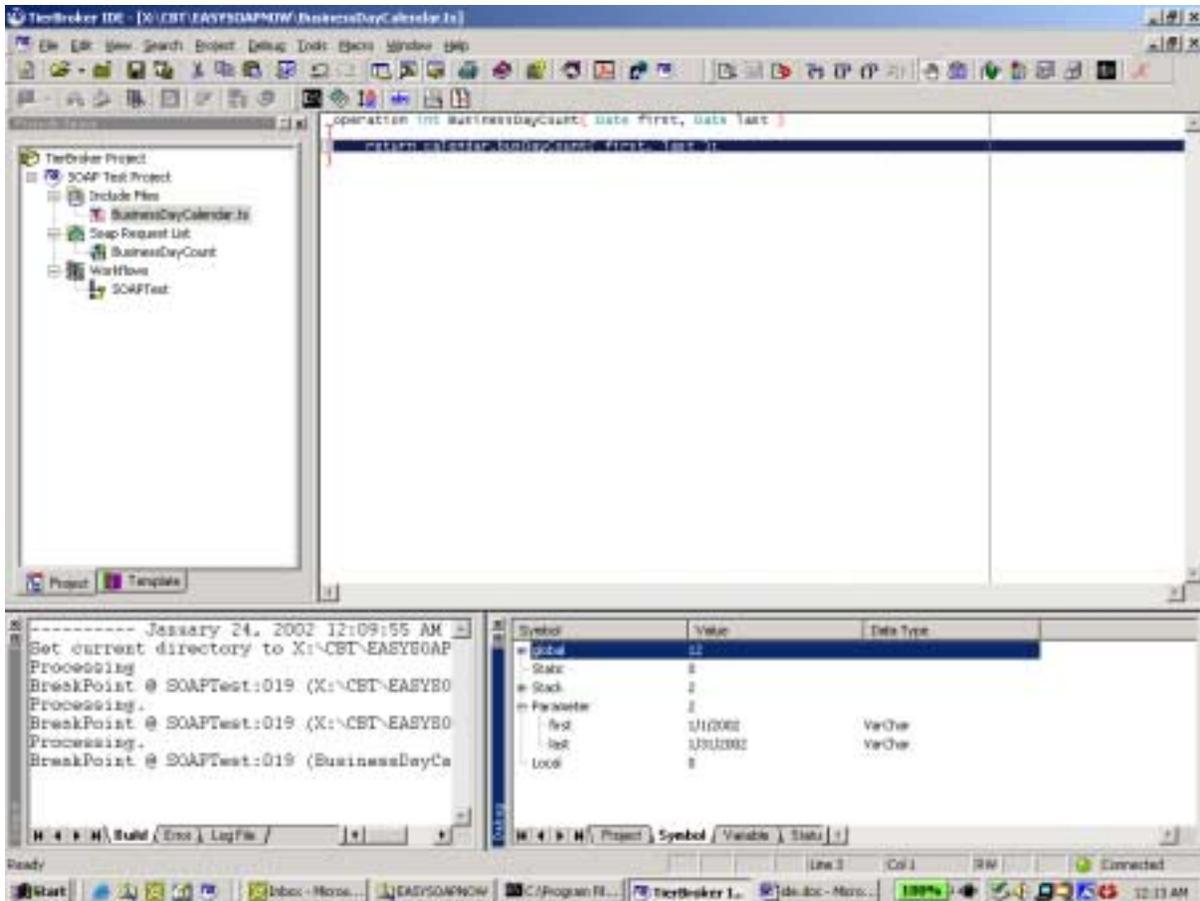
21.7. Debugging the Workflow

After running the workflow we notice a few things have occurred.

1. The execution has stopped on the line of script with a Breakpoint statement.
2. The log window shows that the server is processing.
3. The debugging buttons on the toolbar are enabled



At this point you can use the debugging operations to step through the script. Stepping into the `BusinessDayCount` function brings the debugger into that file.



Here we can step further through the TB Script file. By pressing the Symbols button, a list of the project variable, local variable and the function call stack is displayed.

```

C:\Program Files\UDICo\TierBroker\Tb\bin\tb.exe
tb> 21.         void onOccur( )
22.         {
23.             BreakPoint();
24.             BusinessDayCount( '1/1/2002', '1/31/2002' );
*25.         }
26.         11)</Script>
27.         </OnLoad>
28.
29.         </Workflow>
30. </WorkflowList>
31. </Project>
32.
33. <!-- ***** end of soaptest.udl ***** -->

tb>lwf *, C:\Program Files\UDICo\TierBroker\Tb\bin\tlwf.xml, TRUE
Ok

tb>out

tb>lwf *, C:\Program Files\UDICo\TierBroker\Tb\bin\tlwf.xml, TRUE
Ok

tb>lwf *, C:\Program Files\UDICo\TierBroker\Tb\bin\tlwf.xml, TRUE
Ok

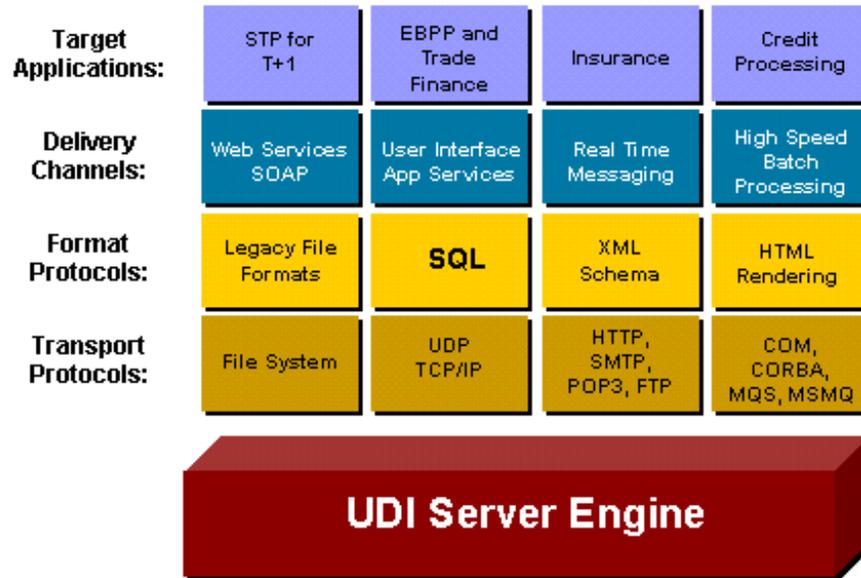
tb>_

```

Looking at the server console you can also see each of the operations and subsequent results that have been logged to the screen.

22. Appendix II: TierBroker Server Architecture

22.1. TierBroker Server Overview



TierBroker's easy adaptability to so many applications on so many platforms comes from the ease with which a developer can mix and match different choices at different levels of the application architecture. When configuring an input or output queue in a TierBroker workflow, you select the appropriate method of data transport (the transport protocol) and the format of the data (the format protocol) appropriate to the workflow source or destination specified as a requirement for your middleware application.

For example, many describe SOAP communication as XML being sent via HTTP, but they don't realize that the SOAP 1.1 specification also provides for sending the XML of a SOAP request or response over SMTP. TierBroker has built-in support for both HTTP and SMTP, making full SOAP compliance much easier for a web service built using TierBroker.

TierBroker also separates the choice of delivery channel from your choice of format protocol and transport protocol. For input, it can read records joined from a multi-table database, watch a directory for the appearance of files, listen to a message queue, watch for real-time input from a data entry operator using web-based HTML forms, and more. For output, it can do all the same things in reverse—in fact, converting a configured input queue to an output queue is often as simple as changing one dialog box setting.

22.2. Development Architecture

The TierBroker IDE lets you assemble TierBroker middleware projects by dragging and dropping icons, writing simple or complex scripts, and setting configuration parameters for the various pre-built components of your middleware. While it is easy to use, it's not designed to turn receptionists into middleware developers; instead, it lets developers who know what they need specify those needs, set the appropriate parameters, and get their middleware up and running as quickly as possible.

When you assemble a middleware project with TierBroker, you're creating and configuring three kinds of things:

- *Object Classes* define the structure of the objects that TierBroker will read from its input queues and write to its output queues. You can enter these using TierBroker's scripting language for complex object definition, but in most cases you can add object class definitions to your project by telling the TierBroker IDE to read in existing data structure definitions such as an XML DTD, an SQL SELECT statement, or an MS Query file.
- *Workflows* tell the TierBroker Server where to get data objects and what to do with them. A typical workflow has at least one input queue, a process to manipulate the data from that input queue, and at least one output queue. If you've developed and tested a complex workflow that reads from and writes to CSV and XML files on your notebook computer's hard disk, small changes to the workflow queue configurations are all you need to run the same complex workflow on a high-end Sun workstation reading from a Sybase database and writing to an MQ Series message queue.
- *Maps* describe how to convert objects of one class into objects of another class. If your output objects have the same structure and field names as your input objects (for example, when converting the rows of an SQL table into XML elements whose subelements have the same names and ordering as the SQL columns), maps are unnecessary, but when you need, them, the TierBroker IDE makes it easy to rename, reorder, and manipulate the fields of one object to create a new one.

The configuration of workflow queues is where you identify the protocols necessary to get and deliver the data routed by TierBroker. By keeping this information completely separate from the map and object class configuration, TierBroker lets you define object classes and maps with no system-level dependencies, which makes them easier to re-use in multiple projects, which leads to faster application development.

UDICo's TierView, in addition to letting you monitor the progress of the TierBroker Server as it performs production workflows, serves as a powerful debugger for applications in development. You can set breakpoints and then, when execution is interrupted, examine data and perform commands at the debugging prompt before resuming execution. For even greater control over your middleware workflows, the TierBroker engine is available as a C++ class library and API.

22.3. Scripting and Multiple Language Support

Business logic associated with workflows is specified using scripts attached to workflows. Routing an order with an invalid customer code to an error queue instead of to its original target is as simple as writing a JavaScript "if" statement. The TierBroker IDE comes with a full-featured text editor that offers context-sensitive help, syntax completion, color-coded keywords, and the ability to define and run your own macros as you create your scripts.

The two default scripting languages supported by TierBroker are ECMAScript (also known by the names of its Netscape implementation, "JavaScript," and its Microsoft implementation, "Jscript") for procedural logic and SQL for easier manipulation of table-oriented data.

This combination of JavaScript and SQL gives you a great deal of control over your logic and your data, but you can use other languages as well. TierBroker's "Schema for Language" technology makes it possible to support an arbitrary number of scripting languages for its scripting; a separate white paper describes this in more detail.

23. Appendix III: The TierBroker Pricing Model

23.1. Open Pricing for Open Systems

The TierBroker product family is easy to use and easy to understand. Our pricing model is based upon feedback that we have received from our institutional clients as well as start-up companies that embed the TierBroker Server as a part of their products and solutions.

Our competitors sell *Enterprise Software*, which means you have no idea what you are actually buying until you see the bill for all of the hidden costs. Application dependencies, custom coding, system integration, consultants, architects – all of these things make you feel like you're starting own software company instead of running your business! Our pricing model is published on the web. We also have programs for OEM, VAR and ISV distributors.

23.2. The TierBroker Product Family

TierBroker Enterprise Server – Available in Windows or selected UNIX operating systems. No limits on users or connections. Priced at **\$30,000 per server domain**, not per CPU. Our institutional clients hate CPU based pricing, because it is too difficult to administer. Our pricing is based upon a *per project*, or *per domain* utilization.

TierBroker Standard Edition – Available under Windows 95-98, NT, 2000, XP. Up to 10 simultaneous connections for real time applications or 10,000 records for single batch processing. Priced at **\$4,500 per server domain**. Ideal for building corporate intranet applications or as an add-on for small to medium sized web sites.

TierBroker Professional Edition – Available under Windows 95-98, NT, 2000, XP. Single user TierBroker Server with one simultaneous connection for real-time applications or 10,000 records for single batch processing. Priced at **\$950 per desktop**. Ideal for building desktop peer-to-peer applications or for integrating real-time data into MS Office using Web Services.

TierBroker Developer – Includes a single copy of the TierBroker Professional Edition and the TierBroker Integrated Development Environment (IDE). The TierBroker IDE is similar to MS Visual Studio and has advanced project editing and debugging features. Priced at **\$1,500 per desktop**. Ideal for developers who are building and testing TierBroker applications.

TierBroker Reader – The read-only version of TierBroker Professional. TB Reader is similar to Adobe Acrobat, because it is a **\$FREE** plug-in that allows client applications to access TierBroker servers and integrate their real-time services into desktop applications. The TB Reader allows clients to **push or pull** data to any TierBroker host.

23.3. The UDICo OEM Program

UDICo is committed to working with integrator partners and independent software vendors. Contact UDICo for information concerning distributor pricing and product licensing.

23.4. UDICo Bonus – The Customer’s Bill of Rights

The following section talks about our competitor’s pricing models and enterprise software in general. It is based upon our experience in consulting.

The following material is provided by UDICo and does not represent the views or policies of any of our investors, partners or clients.

The **Customer’s Bill of Rights** is our belief that you should know exactly what you are buying and exactly what it will cost. The price you pay should be the same price that everyone else pays. The products you buy should function correctly and if they don’t then you should not have to pay to have them fixed.

23.5. What Our Competitors Don’t Want You to Know

At UDICo we are committed to *transparent pricing*. That means there is a list price for our products, which is published in this document and on the web. Most of our competitors do not have open pricing for three very good reasons:

1. They don’t have a product – they have a library of source code that consultants customize for each installation. They can’t say what the price is, because the product they deliver is going to be written at the customer location.
2. They don’t want you to know. They want to create the illusion that their product is so exclusive that their clients will accept any price because of the value they provide.
3. They want to see how much money you have and charge as much as possible.

When you purchase enterprise software, the value you get for your money will depend upon who does the negotiating. Large companies have in-house contract negotiators and lawyers who specialize in managing vendor relationships.

Big institutional clients care *a lot* about pricing and always get the lowest prices. They also get *give-backs* in the form of *most-favored-client* agreements, which means that if the software vendor sells their product for less money to anyone they get an instant refund on the difference.

23.6. How to Negotiate for Enterprise Software

If you do need to buy a *solution* from an enterprise software vendor, here are just a few tips you will want to remember as you negotiate the contract.

- Always start with an RFP. The longer it is, and the more detailed, the more information you will get for free. Once a vendor has *skin in the game*, the more they will try to win the business by discounting their list price and throwing in freebies.
- If you can afford to hire a consultant to manage the RFP and package selection, then you will get better results. The consultant should give you a good deal on the RFP, because they will want to win the implementation services.
- Time is on your side, especially in this economy. The words, “Send it back to legal.” or “Send it back to the steering committee.” are very powerful weapons.

- All consulting rates above \$150/hour can be discounted with very few exceptions. Most vendors will quote rates at \$2,000/day and up. Start the negotiation at \$1,000/day and tell them you have cap on both day rate and expenses.
- Never pay list price. Every salesperson at every company is under pressure to make their quarterly quota. Always check the recent sales history of the vendor. The more sales are down or flat the more they will be willing to offer a discount.
- Always obtain a *most-favored-client* agreement as an attachment to the basic contract. This agreement should state that if the software vendor sells their product for less money to anyone within a specific time horizon (1-3 years is the range), then they will refund you the difference.

These suggestions come from our experience implementing enterprise software packages for global financial institutions. As you put these bullet points on the table, you will see the sales reps turn pale. They will bubble and stammer and try to make you believe that they are above this type of negotiation. Don't believe it for even one second. The more they cry and squirm, the more they can taste the commission. You the customer have all the power – and that fact alone is the real secret to getting the best value for your money.

24. Appendix IV: Installation FAQ and Quick Start

24.1. What is TierBroker?

The TierBroker Server is a middleware product that lets you easily design simple or complex workflows between a wide choice of data resources and then lets you run those workflows at high speed, in real time, on a variety of platforms.

Application Definition

TierBroker is a *server* for middleware *client* applications. It is similar to a SQL server, because it runs independently of its client applications and can execute scripts, which instruct it to perform a specific batch or real time function.

 The TB Server is similar to a SQL server. It executes TierBroker projects instead of SQL commands.

Unlike most SQL servers, TierBroker has been designed for client-side computing, and can be installed on something as small as Windows 95 laptop or as large as a Sun E-10000 UNIX server.

The Role of Middleware

Middleware is glue – it is the software that brings different applications together by translating and transforming data. Middleware is also used to read from and write to different types of databases such as Oracle and Sybase and messaging applications such as MQ Series. The TierBroker Server can be used for both real time and batch processing and has successful deployments in both categories of system interfacing.

24.2. How Do I Get the TierBroker Software?

Go to www.udico.com and complete the registration information on the download page. You can select either TierBroker Server or the complete TierBroker Integrated Development Environment or IDE.

24.3. What Do I Need to Install?

Windows Environment

The most common installation procedure is to download the complete install using `setup.exe`. A license key will be sent through e-mail to unlock features and user levels. The TierBroker Server can also be installed separately as a standalone application or to update a TierBroker installation.

UNIX Environment

The most common installation procedure is to download the `gzip` file for the TierBroker Server. A license key will be sent through e-mail to unlock features and user levels.

24.4. *Where Should I Install TierBroker?*

TierBroker must have TCP/IP level visibility to the files, databases and messaging sockets that it needs to connect to at runtime. A good location for TierBroker is the server that is also hosting the target application.

Windows Environment

Select a computer for TierBroker. In a production environment this will generally be a server located in a data center and accessed using Cytrix, PC Anywhere or Carbon Copy. This should be on the same subnet as the files databases and TCP/IP ports TierBroker must access for its target application.

UNIX Environment

Select a server for TierBroker. This does not need to be the same server as the database server, although this will improve performance (only relevant in very high volume applications). A good location for TierBroker is the same server as the primary application server.

24.5. *What are the Hard Disk Requirements?*

Windows Environment

The full IDE requires approximately 70MB. The full server requires approximately 10MB. The minimum server image is 4MB which is achieved by removing the Computer Based Training (CBT) modules, PDF and HTML documentation files under `tb\bin` and `tb\bin\cmd\english`.

Unix Environment

The full server requires approximately 14MB. The minimum server image is 8MB which is achieved by removing the Computer Based Training (CBT) modules, PDF and HTML documentation files under `tb\bin` and `tb\bin\cmd\english`.

24.6. *What are the Memory Requirements?*

UNIX and Windows Environments

The minimum runtime server image is 16MB. This will vary depending upon the size of the project, number of concurrent sessions and data stored in lookup caches. Runtime images of 100-200MB are not uncommon in production environments.

24.7. *What are the Runtime Dependencies?*

TierBroker has no external dependencies other than certain external shared libraries. Typically, these are associated with the target database, such as Oracle, MS SQL Server or Sybase. Other dependencies can be created at the application level, if a TierBroker project declares and calls external shared libraries.

 TierBroker is great middleware because it has no dependencies. It can work equally well under MS Windows as Solaris.

Note that applications that use ODBC under Windows, will need to configure ODBC connections.

25. Appendix V: Installation of TierBroker for Windows

25.1. *Summary of Installation the Procedure*

Download the Software

Download the TierBroker software from the UDICo website.

Install the Software

Run the installation program or unzip the server software.

Install the License Key

Install the TierBroker Server and/or IDE license keys.

25.2. *TierBroker Components*

The TierBroker Server

The TierBroker Server is the middleware engine. It is a console driven program that has the same operational procedures under Windows and UNIX.

The TierBroker IDE

The TierBroker Integrated Development Environment, or IDE, is a programmer's text editor and debugging tool for TierBroker project applications.

 The TierBroker IDE is not required for TierBroker development. Any text editor, including *vi*, can be used to edit TierBroker project files and source code.

Note that the TierBroker IDE is available under Windows only.

The Kdb In-Memory Database

Kdb is an ultra-high performance database and functional programming language. For information on Kdb refer to www.kx.com for documentation. Kdb is available in the Trial Version and Enterprise Version only.

25.3. *Installing Updates*

Updates can be downloaded and installed at any time. License files are not overwritten by the installation process. If changes are made to default configuration files, these should be backed up and restored before performing the update process.

25.4. Download Installation Software

The current version of the TierBroker software is available at www.udico.com. The download you select will depend upon your installation requirements.

Complete Development System

To install a complete development system, select the Windows `setup.exe`, which will use Install Shield to unpack the TierBroker Server and IDE.

TierBroker Server for Windows

The TierBroker server for Windows is available as a separate download. The TierBroker server zip uses the following file naming conventions:

```
tb-V411.exe // The TierBroker Server Version 4.11
```

The TB Server can be downloaded separately if the complete installation is not required.

The Kdb In-Memory Database

The Kdb database from Kx Systems (www.kx.com) is installed with the Trial Version of TierBroker, and is licensed for use with the Enterprise Version of the TierBroker Server.

25.5. Obtaining License Keys for the TierBroker Server and IDE

The TierBroker license is obtained by e-mail. At the time TierBroker is purchased license keys are e-mailed to the customer location for installation in the TierBroker runtime environment.

☞ If the license file is lost or becomes corrupted contact UDICo customer support to request a replacement.

The license key should be installed after the most recent copy of the TierBroker software has been downloaded and installed from the UDICo web site.

25.6. TierBroker Server and IDE Licensing

There is only one download version of each TierBroker component. The difference between the Evaluation Version of TierBroker and the licensed non-expiring version is the presence of the software license key.

☞ TierBroker does not use a license daemon. The license key contains information regarding end user license permissions.

The TB Server and IDE use similar methods to unlock the software, but they require separate keys, since the server can also be deployed under UNIX.

The TierBroker Server License Key

The TierBroker Server license is controlled by the `license.xml` file. This file must be visible to the server at runtime to enable the software license.

```
c:\Program Files\UDICo\TierBroker\tb\bin
```

The TierBroker Server runtime directory is the recommended location for the license.xml file, although any directory in the Server load path can be used.

The TierBroker IDE

The TierBroker IDE license is controlled by a license utility, which contains the end user registration information. The license utility modifies the IDE installation to become non-expiring.

25.7. Installation of the TierBroker IDE with Setup.exe

From the Windows Start menu run the `setup.exe` file and follow the instructions in Install Shield.

Setup.exe installs the complete environment, including the TierBroker Server and Kdb runtime components.

After the installation is complete, the TierBroker IDE can be accessed from the Start-Programs sub menu in Windows. The TierBroker Server can be run from the IDE or from the DOS prompt.

25.8. Installation of TierBroker Server with TB-VNNN.exe

If the TierBroker Server is being installed separately, then the self-extracting zip file can be run from the Windows console or file manager.

```
c:\Program Files\UDICo\TierBroker\tb
```

This is the default installation path for the TierBroker Server under Windows.

Any location can be used for the TierBroker server. Modify the Path environment variable to include the server binary subdirectory `tb/bin`.

Caution should be used to ensure that only one copy of the TierBroker Server is active in the executable path or inconsistent results be obtained by invoking different server versions.

25.9. Modification of the Executable PATH

Windows 95-98

Modify the `autoexec.bat` file PATH variable to include TierBroker.

```
PATH=%PATH%;c:\progra~1\UDICo\TierBroker\tb\bin
```

The computer must be restarted for this change to become effective.

Windows NT, 2000, XP

Use the System Administrator application to modify the PATH environment variable.

25.10. Installation of the Excel Add-in

The Excel Add-in for Web Services is installed as a component of the full TierBroker Developer package. If the TierBroker Server is installed separately without the IDE, then the Excel Add-in must be installed using the following procedure.

1. Start Excel.
2. Select **Tools-Add-Ins** from the main menu.
3. Select **Browse** and choose `\Program Files\tb\cbt\Soap\UDICo.xla`

These steps install the TierBroker Web Services Add-in for MS Excel.

25.11. Installation of the Kdb Database Engine

The Kdb database engine can be updated by downloading the Kdb installation program from www.kx.com.

26. UDICo Contact Information

26.1. *Company Overview*

The Universal Data Interface Corporation is a technology spin-off from PricewaterhouseCoopers LLP. The founder of the company, Adam Greissman, is specialist in designing and implementing globally distributed transaction processing systems and a pioneer in the development of XML specifications for financial transactions. The TierBroker is a middleware product targeted at EAI and B2B applications, and has been used to implement systems for financial institutions, beginning with the Chase Manhattan Bank in 1993. The TierBroker has been deployed for applications such as front-to-back office reconciliation, credit and market risk management (Sungard and Reuters), ERP integration for SAP and PeopleSoft, and CRM applications such as BroadVision and Siebel.

26.2. *Contact Information*

Questions about this document may be addressed to the following UDICo officers:

Adam Greissman
Chief Executive Officer
1.917.497.4342 – alg@UDICo.com

Simon Smith
Chief Financial Officer
1.646.387.9600 – simon.smith@UDICo.com

Universal Data Interface Corporation
95 Wall Street, 21st Floor
New York, NY 10005
Fax: 212.607.7580