



ASTA SOAP

© 1997-2002 Asta Technology Group, Inc.

Table of Contents

Part I SOAP Introduction	4
1 Asta SOAP Support	4
Soap Components	6
ServerSide	6
TAstaHTTPListener	7
Properties	7
Active	7
Port	7
Realm	7
SessionCount	7
SOAPServer	8
SocketServer	8
Events	8
OnConnect	8
OnDisconnect	8
OnError	8
OnExecute	8
OnLogin	9
TAstaSOAPMethod	9
Properties	9
Documentation	10
InputParams	10
OutputParams	11
Method	11
Server	11
Methods	11
DoAction	11
Events	11
OnAction	11
TAstaSoapMethodElement	12
TAstaSOAPMethodOwner	12
Methods	12
AddMethod	12
Count	12
DeleteMethod	12
GetMethod	12
IndexOf	12
TAstaSOAPServer	13
Properties	13
ResponseEncoding	13
Methods	13
Execute	13
TAstaSOAPSession	13
Properties	14
Params	14
Request	14
Response	14

Result	14
ClientSide.....	14
TAstaHTTPConnection.....	14
Properties	15
Host	15
KeepAlive	15
Method	15
Page	15
Password	16
Port	16
ProxyHost	16
ProxyPort	16
RequestData	16
RequestHeaders.....	16
ResponseCode.....	16
ResponseData	16
ResponseMessage.....	17
UserName	17
UseSSL	17
Methods	17
Abort	17
Close	17
Execute	17
Events	17
OnConnect	18
OnDisconnect	18
OnRequest	18
OnResponse	18
TAstaSOAPClient	18
Properties	18
Connection	19
Method	19
MethodURI	19
Params	19
Request	19
RequestEncoding.....	19
Response	19
Result	20
SOAPAction	20
Methods	20
Execute	20
Events	20
OnComplete	20
OnShowRequest.....	20
OnShowResponse	20
Common	21
TAstaSOAPPParams.....	21
Properties	21
AsBase64Binary.....	21
AsBoolean	21
AsDateTime	21
AsDecimal	22
AsDouble	22
AsDuration	22

AsHexBinary	22
AsInteger	22
AsString	22
Attributes	22
DataType	23
Name	23
Namespace	23
Methods	23
Add	23
Clear	23
Count	23
Delete	24
Exists	24
Get	24
GetFullName	24
IndexOf	24
2 Soap Examples	24
ASTA Soap Example Server	24
StringReverse.....	26
ArraySum.....	27
Distance	27
ASTA Soap Example 'Clients	27
Component Example.....	27
Non Delphi Client Examples	28
Visual Basic.....	28
Asta SOAP Explorer	28
3 AstaIO	29
4 ASTA 3 Soap Support	30

1 SOAP Introduction

Web Services are changing the Internet Landscape. A set of standards has finally been developed that allow for application logic to be executed over the Internet regardless of language and operating system. SOAP (Simple Object Access Protocol) is basically XML over HTTP and the SOAP standard IS the basis for WebServices.

ASTA provides for cross platform SOAP Service Support allowing SOAP Services to be implemented on the server on Windows or Linux and for ASTA SOAP Clients to execute those SOAP Services from Win32, Linux, WinCE, Palm and Java clients.

ASTA SOAP technology has NO runtime or server fees and is not dependent on ASTA 3 or AstaIO servers. ASTA 3 and AstaIO integration is supported but non ASTA users can use ASTA SOAP technology and code servers any way they want!

[ASTA SOAP Support](#)

[ASTA Soap Explorer](#)

[ASTA Soap Amazon Client Example](#)

Links for more Information on SOAP

<http://www.develop.com/soap/>

<http://static.userland.com/xmlRpcCom/soap/SOAPv11.htm>

<http://www.w3.org/TR/soap12-part1/>

<http://msdn.microsoft.com/msdnmag/issues/0300/soap/soap.asp>

1.1 Asta SOAP Support

ASTA provides SOAP support at a number of levels.

On the first level, SOAP Servers can be created for use in ISAPI DLL that have no dependency on any ASTA transport. Neither ASTA 3 nor AstaIO is required. SoapMethods are easily created by filling in some simple properties or invoking a couple of property editors and then coding an [OnAction](#) Event.

Soap Clients can easily be written in Delphi by just filling in a couple of properties on an [TAstaSoapClient](#) and calling [Execute](#).

SOAP Servers in ISAPI DLL's	TAstaSoapServer TAstaSoapMethod
Stand alone HTTP SOAP Server	TAstaSoapServer TAstaSoapMethod TAstaHTTPListener
SOAP Clients	TAstaSoapClient TAstaHTTPConnection

Integration with ASTA 3 and AstaIO Servers is available also

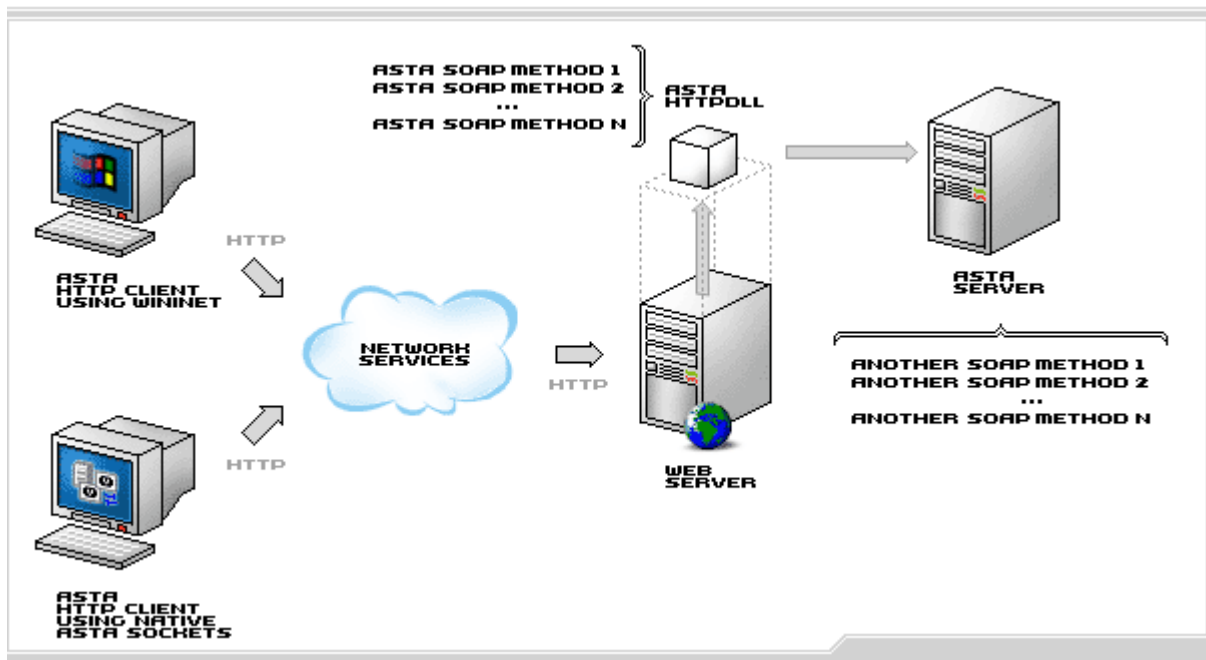
ASTA 3 Integration	Ties TAstaSOAPMethods to ASTA 3 ServerMethods
AstaIO Integration	Ties TAstaSOAPMethods to TAstaIOServerExecMethods

Server Side

TAstaSoapMethods allow for developers to define soap Methods that can be used directly in Isapi DLL's, or Apache DSO's with Kylix. Additionally SOAP methods can be defined in ASTA Servers. ASTA Server ServerMethods have a boolean property of SoapService: Boolean. If this is true, then that servermethod can be published as a SOAP Service. **ASTA HTTP Server, AstaISAPI, AstaDSO** ASTA Soap Servers can be run as a Standalone SOAP Server or as an ISAPI DLL or Apache DSO. When running standalone, that soap server, which uses the AstaIO Native Cross platform sockets (available in ASTA 3 also), can publish SOAP services coded directly from the ASTA HTTP Server or through ASTA Servers.

When the AstaHTTPSoapServer is started up, a list of available ASTA Servers is made available and the SOAP Server queries the ASTA Servers for available ServerMethods tagged as being available to be published as a SOAP Service. The SoapServer then publishes a WSDL file that contains the sum of all of the ServerMethods coded directly from the SOAP Server and from ASTA Servers. Thus soap services can be made available with no knowledge of XML. When using ASTA Servers, all threading and database connection pooling is handled by the ASTA servers so that this solution is extremely scalable. Load Balancing and Fail Over options are available also. The AstaSkyWire line will include the ability for scripted ASTA server methods to be defined via VBScript. Once defined, those methods will flow through the normal ASTA SOAP architecture. Thus SOAP Services can be defined at runtime without re-starting ASTA Servers or SOAP Servers. There will also be an ASTA COM Proxy server that can read COM Methods and publish those methods as SOAP Servers.

Client Side The whole point of SOAP services is that any client application can call any SOAP Service regardless of operating system or development tool used. So of course ASTA Soap Services can be called by any SOAP Client. ASTA 3 and AstaIO also provide a native XML parser written in Pascal. This XML parser is used with the TAstaSOAPClient to provide SOAP Clients under Delphi and kylix. The XML Parser has been translated to portable C++ and will be available on Palm, WinCE, and Linux allowing Pda clients to call SOAP Services Directly. Additionally, ASTA clients will be able to call SkyWire servers that can act as a SOAP Proxy. This means that Palm or WinCE clients can access SOAP services without the need for an XML parser but through normal ASTA messaging calls.



1.1.1 Soap Components

ASTA provides components to deploy SOAP Services on SOAP Servers or to consume SOAP services from Delphi Clients. The Core ASTA Soap components have nothing to do with ASTA 3 or AstaIO but can be used by themselves to deploy scalable SOAP services or thin, fast SOAP Clients. ASTA 3 and AstaIO servers can also be integrated with ASTA SOAP Technology but there are no dependencies on Asta Code, except for the AstaIO cross platform sockets that are used in ASTA SOAP Standalone HTTP Server support.

Server Side

Client Side

1.1.1.1 ServerSide

TAstaHttpListener
 TAstaSoapMethod
 TAstaSoapMethodOwner
 TAstaSoapServer

1.1.1.1.1 TAstaHTTPListener

[Properties](#) : [Events](#)

Unit

AstaHTTPListener

Declaration

```
TAstaHTTPListener = class(TComponent)
```

Description

ASTA allows for SOAP Servers to be created in an ISAPI dll or a StandAlone HTTP Server. The TAstaHttpListener uses the cross platform AstaIO sockets to implement a standalone HTTP Server for SOAP Services.

Attach a TAstaHTTPListener to your TAstaSoapServer to listen for and receive HTTP SOAP requests on the default port of 80. Set the Active property to True to begin listening for requests on the TCP/IP port specified by the Port property.

1.1.1.1.1.1 Properties

[TAstaHTTPListener](#)

[Active](#)

[Port](#)

[Realm](#)

[SessionCount](#)

[SOAPServer](#)

[SocketServer](#)

[TAstaHTTPListener](#)

Declaration

```
property Active: Boolean;
```

Description

Set Active to True to allow TAstaHTTPListener to listen for and process SOAP requests over HTTP. Set Active to False to stop processing requests.

[TAstaHTTPListener](#)

Declaration

```
property Port: Integer;
```

Description

[TAstaHTTPListener](#)

Declaration

```
property Realm: String;
```

Description

[TAstaHTTPListener](#)

Declaration

```
property SessionCount: Integer;
```

Description

[TAstaHTTPListener](#)

Declaration

```
property SOAPServer: TAstaSOAPServer;
```

Description

[TAstaHTTPListener](#)

Declaration

```
property SocketServer: TAstaSocketServer;
```

Description

1.1.1.1.2 Events

[TAstaHTTPListener](#)

[OnConnect](#)

[OnDisconnect](#)

[OnError](#)

[OnExecute](#)

[OnLogin](#)

[TAstaHTTPListener](#)

Declaration

```
property OnConnect: TNotifyEvent;
```

Description

[TAstaHTTPListener](#)

Declaration

```
property OnConnect: TNotifyEvent;
```

Description

[TAstaHTTPListener](#)

Declaration

```
property OnError: TAstaHTTPError;
```

```
TAstaHTTPError = procedure(Sender: TObject; ErrorStr: string) of object;
```

Description

[TAstaHTTPListener](#)

Declaration

```
property OnExecute: TAstaHTTPExecute;
```

```
TAstaHTTPExecute = procedure(Sender: TObject; Headers: TStrings; Request:
```

```
string; var Response: string) of object;
```

Description

[TAstaHTTPListener](#)

Declaration

```
property OnLogin: TAstaHTTPLogin;  
TAstaHTTPLogin = procedure(Sender: TObject; Headers: TStrings; var Accept:  
    Boolean) of object;
```

Description

1.1.1.1.2 TAstaSOAPMethod

[Properties](#) : [Methods](#)

Unit

AstaSoapMethod

Declaration

```
TAstaSoapMethod = class(TComponent)
```

Description

TAstaSoapMethod is used to implement a SOAP service on an SOAP Server. It has no ties to ASTA 3 or AstaIO and can be deployed in an ISAPI dll or a standalone SOAP Server. A [Method is Named](#), and [InputParams](#) and [OutPutParams](#) are defined. Then some code is written in the [OnAction](#) event.

[Example Server Side SOAP Methods](#)

Note: Since SOAP does not support result sets, ServerMethods can only be used that have no result sets and only use parameters. DataSets can be passed as a parameter if you do need to bring back a result set to your soap client.

1.1.1.1.2.1 Properties

[TAstaSOAPMethod](#)

[Documentation](#)

[InputParams](#)

[Method](#)

[OutputParams](#)

[Server](#)

[TastaSOAPMethod](#)

Declaration

property Documentation: **string**;

Description

Used for comments to describe the SOAP service.

[TastaSOAPMethod](#)

Declaration

property InputParams: [TastaSoapMethodElement](#)

Description

SOAP Services can have any number of Params. This InputParams Property editor, below allows you to add params.

Name	Type
Incoming	String

Name: Incoming

Simple type: String

Complex type:

[TAstaSOAPMethod](#)**Declaration**

property OutputParams: TAstaSoapMethodElement

[TAstaSOAPMethod](#)**Declaration**

property Method: **string**;

Description

This is the name that will be used by remote clients to invoke the SOAP Service.

[TAstaSOAPMethod](#)**Declaration**

property Server: [TAstaSoapMethodOwner](#);

Description

1.1.1.1.2.2 Methods

[TAstaSOAPMethod](#)[DoAction](#)[TAstaSOAPMethod](#)**Declaration**

procedure DoAction(Session: [TAstaSoapSession](#));

Description

1.1.1.1.2.3 Events

[TAstaSOAPServer](#)[OnAction](#)[TAstaSOAPMethod](#)**Declaration**

property OnAction: TSoapActionEvent;
TSoapActionEvent = **procedure**(Sender: TObject; Session: [TAstaSoapSession](#)) **of**
object;

Description

1.1.1.1.3 TAstaSoapMethodElement

Enter topic text here.

1.1.1.1.4 TAstaSOAPMethodOwner

[Methods](#)**Unit**

AstaSoapMethod

Declaration

```
TAstaSoapMethodOwner = class (TComponent)
```

Description

1.1.1.1.4.1 Methods

[TAstaSoapMethodOwner](#)[AddMethod](#)[Count](#)[DeleteMethod](#)[GetMethod](#)[IndexOf](#)[TAstaSoapMethodOwner](#)**Declaration**

```
procedure AddMethod(Method: TAstaSoapMethod);
```

Description[TAstaSoapMethodOwner](#)**Declaration**

```
function Count: Integer;
```

Description[TAstaSoapMethodOwner](#)**Declaration**

```
procedure DeleteMethod(Method: TAstaSoapMethod);
```

Description[TAstaSoapMethodOwner](#)**Declaration**

```
function GetMethod(Name: string): TAstaSoapMethod; overload;
```

Description[TAstaSoapMethodOwner](#)

Declaration

```
function IndexOf(Name: string): Integer;
```

Description

1.1.1.1.5 TAstaSOAPServer

[Properties](#) : [Methods](#) : [Events](#)

Unit

AstaSoapServer

Declaration

```
TAstaSoapServer = class(TAstaSOAPMethodOwner)
```

Description

1.1.1.1.5.1 Properties

[TAstaSOAPServer](#)

[ResponseEncoding](#)

[TAstaSOAPServer](#)

Declaration

```
property ResponseEncoding: string;
```

Description

1.1.1.1.5.2 Methods

[TAstaSOAPServer](#)

[Execute](#)

[TAstaSOAPServer](#)

Declaration

```
procedure Execute(Request: string; var Response, Error: string);
```

Description

1.1.1.1.6 TAstaSOAPSession

[Properties](#)

Unit

AstaSoapMethod

Declaration

```
TAstaSoapSession = class(TObject)
```

Description

1.1.1.1.6.1 Properties

[TAstaSoapSession](#)

[Params](#)
[Request](#)
[Response](#)
[Result](#)

[TAstaSoapSession](#)

Declaration

property Params: [TAstaSoapParams](#);

Description

[TAstaSoapSession](#)

Declaration

property Request: [TAstaSoapParams](#);

Description

[TAstaSoapSession](#)

Declaration

property Response: [TAstaSoapParams](#);

Description

[TAstaSoapSession](#)

Declaration

property Result: [TAstaSoapParams](#);

Description

1.1.1.2 ClientSide

TAstahttpConnection
TAstaSoapClient

1.1.1.2.1 TAstaHTTPConnection

[Properties](#) : [Methods](#) : [Events](#)

Unit

AstaHTTPConnection

Declaration

```
TAstaHTTPConnection = class(TComponent)
```

Description

The AstaHTTPConnection component provides the transport to communicate with remote soap services. Internally it uses the Microsoft WinINet.dll so that any proxy settings set in Internet Explorer are available in order to traverse any firewall.

1.1.1.2.1.1 Properties

[TAstaHTTPConnection](#)

[Host](#)

[KeepAlive](#)

[Method](#)

[Page](#)

[Password](#)

[Port](#)

[ProxyHost](#)

[ProxyPort](#)

[RequestData](#)

[RequestHeaders](#)

[ResponseCode](#)

[ResponseData](#)

[ResponseMessage](#)

[UserName](#)

[UseSSL](#)

[TAstaHTTPConnection](#)

Declaration

```
property Host: string;
```

Description

[TAstaHTTPConnection](#)

Declaration

```
property KeepAlive: Boolean;
```

Description

[TAstaHTTPConnection](#)

Declaration

```
property Method: string;
```

Description

[TAstaHTTPConnection](#)

Declaration

```
property Page: string;
```

Description

[TAstaHTTPConnection](#)

Declaration

```
property Password: string;
```

Description

[TAstaHTTPConnection](#)

Declaration

```
property Port: Integer;
```

Description

[TAstaHTTPConnection](#)

Declaration

```
property ProxyHost: string;
```

Description

[TAstaHTTPConnection](#)

Declaration

```
property ProxyPort: Integer;
```

Description

[TAstaHTTPConnection](#)

Declaration

```
property RequestData: string;
```

Description

[TAstaHTTPConnection](#)

Declaration

```
property RequestHeaders: TStrings;
```

Description

[TAstaHTTPConnection](#)

Declaration

```
property ResponseCode: Integer;
```

Description

[TAstaHTTPConnection](#)

Declaration

```
property ResponseData: string;
```

Description[TAstaHTTPConnection](#)**Declaration**

```
property ResponseMessage: string;
```

Description[TAstaHTTPConnection](#)**Declaration**

```
property UserName: string;
```

Description[TAstaHTTPConnection](#)**Declaration**

```
property UseSSL: Boolean;
```

Description

1.1.1.2.1.2 Methods

[TAstaHTTPConnection](#)[Abort](#)[Close](#)[Execute](#)[TAstaHTTPConnection](#)**Declaration**

```
procedure Abort;
```

Description[TAstaHTTPConnection](#)**Declaration**

```
procedure Close;
```

Description[TAstaHTTPConnection](#)**Declaration**

```
procedure Execute;
```

Description

1.1.1.2.1.3 Events

[TAstaHTTPConnection](#)

[OnConnect](#)
[OnDisconnect](#)
[OnRequest](#)
[OnResponse](#)

[TAstaHTTPConnection](#)

Declaration

```
property OnConnect: TNotifyEvent;
```

Description

[TAstaHTTPConnection](#)

Declaration

```
property OnDisconnect: TNotifyEvent;
```

Description

[TAstaHTTPConnection](#)

Declaration

```
property OnRequest: TOnAstaHTTPTrace;  
TOnAstaHTTPTrace = procedure(Sender: TObject; Data: string) of object;
```

Description

[TAstaHTTPConnection](#)

Declaration

```
property OnResponse: TOnAstaHTTPTrace;  
TOnAstaHTTPTrace = procedure(Sender: TObject; Data: string) of object;
```

Description

1.1.1.2.2 TAstaSOAPClient

[Properties](#) : [Methods](#) : [Events](#)

Unit

AstaSoapClient

Declaration

```
TAstaSoapClient = class(TComponent)
```

Description

1.1.1.2.2.1 Properties

[TAstaSOAPClient](#)

[Connection](#)
[Method](#)
[MethodURI](#)
[Params](#)
[Request](#)

[RequestEncoding](#)
[Response](#)
[Result](#)
[SOAPAction](#)

[TAstaSOAPClient](#)

Declaration

property Connection: [TAstaHTTPConnection](#);

Description

[TAstaSOAPClient](#)

Declaration

property Method: **string**;

Description

[TAstaSOAPClient](#)

Declaration

property MethodURI: **string**;

Description

[TAstaSOAPClient](#)

Declaration

property Params: [TAstaSoapParams](#);

Description

[TAstaSOAPClient](#)

Declaration

property Request: [TAstaSoapParams](#);

Description

[TAstaSOAPClient](#)

Declaration

property RequestEncoding: **string**;

Description

[TAstaSOAPClient](#)

Declaration

property Response: [TAstaSoapParams](#);

Description

[TAstaSOAPClient](#)**Declaration**

property Result: [TAstaSoapParams](#);

Description[TAstaSOAPClient](#)**Declaration**

property SOAPAction: **string**;

Description

1.1.1.2.2.2 Methods

[TAstaSOAPClient](#)[Execute](#)[TAstaSOAPClient](#)**Declaration**

procedure Execute;

Description

1.1.1.2.2.3 Events

[TAstaSOAPClient](#)[OnComplete](#)[OnShowRequest](#)[OnShowResponse](#)[TAstaSOAPClient](#)**Declaration**

property OnComplete: TNotifyEvent;

Description[TAstaSOAPClient](#)**Declaration**

property OnShowRequest: TOnAstaSoapTrace;

TOnAstaSoapTrace = **procedure**(Sender: TObject; Data: **string**) of object;

Description[TAstaSOAPClient](#)**Declaration**

property OnShowResponse: TOnAstaSoapTrace;

TOnAstaSoapTrace = **procedure**(Sender: TObject; Data: **string**) of object;

Description**1.1.1.3 Common**

Enter topic text here.

1.1.1.3.1 TAstaSOAPPARAMS

[Properties](#) : [Methods](#)

Unit

AstaSoapParams

Declaration

```
TAstaSoapParams = class(TObject)
```

Description

1.1.1.3.1.1 Properties

[TAstaSOAPPARAMS](#)

[AsBase64Binary](#)

[AsBoolean](#)

[AsDateTime](#)

[AsDecimal](#)

[AsDouble](#)

[AsDuration](#)

[AsHexBinary](#)

[AsInteger](#)

[AsString](#)

[Attributes](#)

[DataType](#)

[Name](#)

[NameSpace](#)

[TAstaSOAPPARAMS](#)

Declaration

```
property AsBase64Binary: string;
```

Description

```
property AsBoolean: Boolean;
```

Description

[TAstaSOAPPARAMS](#)

Declaration

[TAstaSOAPPARAMS](#)

Declaration

```
property AsDateTime: TDateTime;
```

Description

[TAstaSOAPPparams](#)**Declaration**

```
property AsDecimal: Currency;
```

Description[TAstaSOAPPparams](#)**Declaration**

```
property AsDouble: Double;
```

Description[TAstaSOAPPparams](#)**Declaration**

```
property AsDuration: TXDuration;
```

Description[TAstaSOAPPparams](#)**Declaration**

```
property AsHexBinary: string;
```

Description[TAstaSOAPPparams](#)**Declaration**

```
property AsInteger: Integer;
```

Description[TAstaSOAPPparams](#)**Declaration**

```
property AsString: string;
```

Description[TAstaSOAPPparams](#)**Declaration**

```
property Attributes: TAstaSoapParamAttributes;
```

Description

[TAstaSOAPPparams](#)**Declaration**

```
property DataType: TXmlDatatype;
```

```
TXmlDatatype = (  
    xdString, xdBoolean, xdInteger, xdDecimal, xdFloat, xdDouble, xdDuration,  
    xdDateTime, xdTime, xdDate, xdYearMonth, xdYear, xdMonthDay, xdDay,  
    xdMonth, xdHexBinary, xdBase64Binary, xdAnyURI, xdQName, xdNOTATION,  
    xdUnknown, xdStruct, xdArray);
```

Description[TAstaSOAPPparams](#)**Declaration**

```
property Name: string;
```

Description[TAstaSOAPPparams](#)**Declaration**

```
property Namespace: string;
```

Description

1.1.1.3.1.2 Methods

[TAstaSOAPPparams](#)[Add](#)[Clear](#)[Count](#)[Delete](#)[Exists](#)[Get](#)[GetFullName](#)[IndexOf](#)[TAstaSOAPPparams](#)**Declaration**

```
function Add: TAstaSoapParams;
```

Description[TAstaSOAPPparams](#)**Declaration**

```
procedure Clear;
```

Description[TAstaSOAPPparams](#)

Declaration

```
function Count: Integer;
```

Description

[TAstaSOAPPParams](#)

Declaration

```
procedure Delete(Index: Integer);
```

Description

[TAstaSOAPPParams](#)

Declaration

```
function Exists(const Name: string): Boolean;
```

Description

[TAstaSOAPPParams](#)

Declaration

```
function Get(const Name: string; CreateIfNotExists: Boolean = True):  
    TAstaSoapParams; overload;
```

[TAstaSOAPPParams](#)

Declaration

```
function GetFullName: string;
```

Description

[TAstaSOAPPParams](#)

Declaration

```
function IndexOf(const Name: string): Integer;
```

Description

1.2 Soap Examples

ASTA Soap components can be used to create servers or clients. Servers can be deployed as standalone http servers or ISAPI dll.

1.2.1 ASTA Soap Example Server

ASTA example servers are included as an isapi dll or a standalone http server. The server implements 3 methods as examples.

[StringReverse](#)

Takes a string input and returns a string with the characters in reverse order as a result param

[ArraySum](#)

Shows how to use Complex Types in SOAP Methods in this case using an Array of Integers

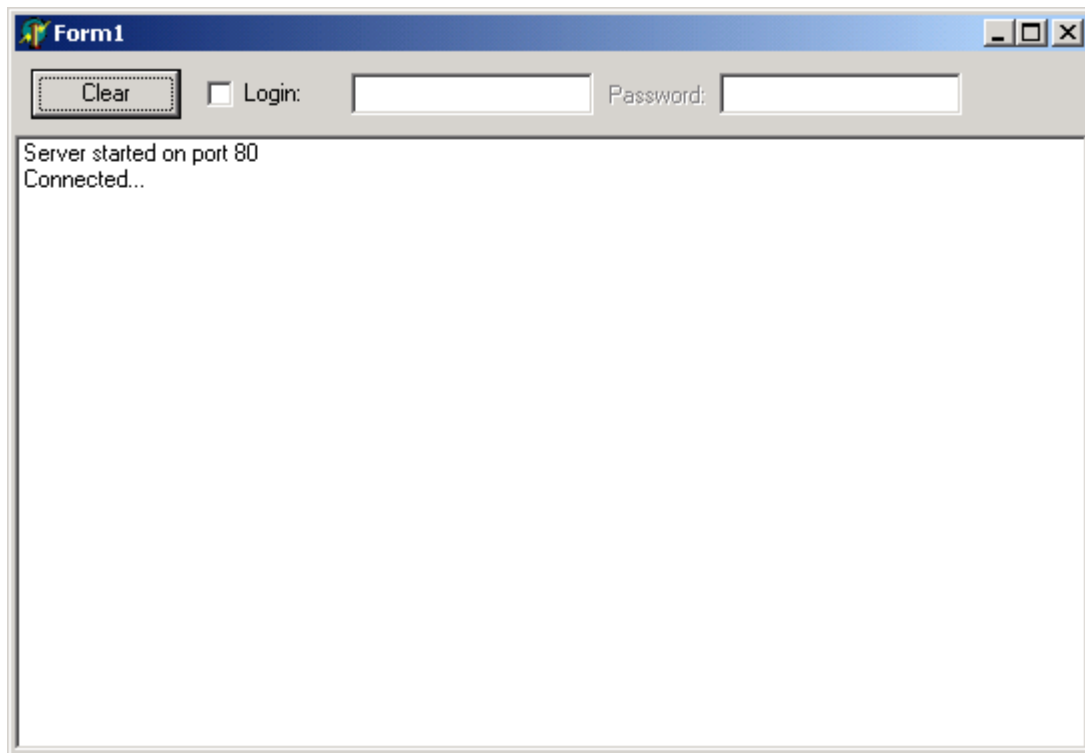
[Distance](#)

Shows how to use Complex Types in calculating Distances taking 2 points as input and returning the distance

Below is a screen shot of a DataModule on an ASTA SOAP Server that shows the methods used in the examples. Both the ISAPI example and stand alone http server share the same DataModule.



Below is a screen shot of an Asta Soap Standalone http server listening for requests



Form1

Clear Login: Password:

Server started on port 80
Connected...

1.2.1.1 StringReverse

```
procedure TSoapData.mtdStringReverseAction(Sender: TObject;  
    Session: TAstaSoapSession);  
var  
    C: Char;  
    Str: String;  
    I, J: Integer;  
begin  
    Str := Session.Params.Get('Incoming').AsString;  
    I := 1;  
    J := Length(Str);  
    while I < J do  
        begin  
            C := Str[I];  
            Str[I] := Str[J];  
            Str[J] := C;  
            Inc(I);  
            Dec(J);  
        end;  
    Session.Result.Get('Result').AsString := Str;  
end;
```

1.2.1.2 ArraySum

```

procedure TSoapData.mtdArraySumAction(Sender: TObject;
  Session: TAstaSoapSession);
var
  I, Result: Integer;
  Numbers: TAstaSoapParams;
begin
  Result := 0;
  Numbers := Session.Params.Get('Numbers');
  for I := 0 to Numbers.Count - 1 do
    Inc(Result, Numbers.Get(I).AsInteger);
  Session.Result.Get('Result').AsInteger := Result;
end;

```

1.2.1.3 Distance

```

procedure TSoapData.mtdDistanceAction(Sender: TObject;
  Session: TAstaSoapSession);
var
  X1, X2, Y1, Y2: Integer;
begin
  X1 := Session.Params.Get('A').Get('X').AsInteger;
  Y1 := Session.Params.Get('A').Get('Y').AsInteger;
  X2 := Session.Params.Get('B').Get('X').AsInteger;
  Y2 := Session.Params.Get('B').Get('Y').AsInteger;
  Session.Result.Get('Result').AsFloat := Sqrt(Sqr(X2 - X1) + Sqr(Y2 - Y1));
end;

```

1.2.2 ASTA Soap Example 'Clients

Enter topic text here.

1.2.2.1 Component Example

This example requires the [ASTA Simple Soap Server](#) to be running either as an ISAPI DLL or as a standalone HTTP server.

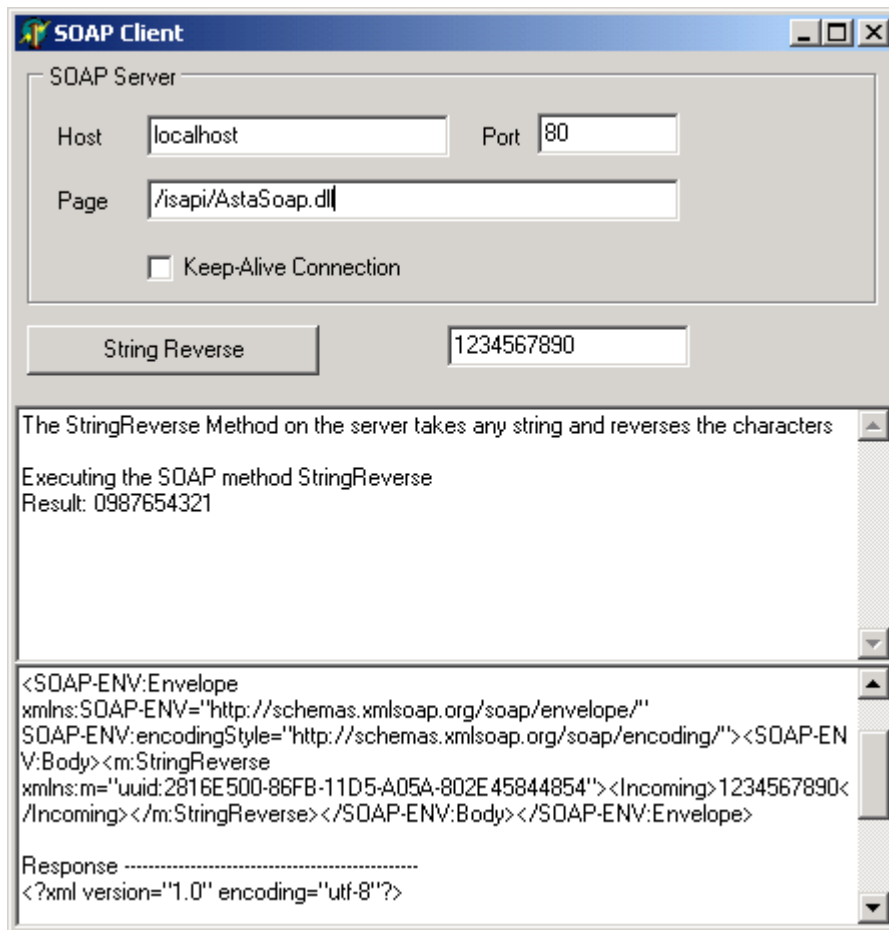
Using the TAstaSoapClient is pretty straightforward. It has a Connection: [TAstaHTTPConnection](#) property that must be setup to point to the correct Soap Server and a [Method](#) that must be set to execute the desired method on the server. In this example the Method is set to [StringReverse](#). Params are set, and then the [Execute](#) method is invoked with any returning Params then available.

Below is some sample code that shows how this is done.

```

procedure TForm1.Button3Click(Sender: TObject);
begin
  Client.Params.Get('Incoming').AsString := StringEdit.Text; ;
  Client.Execute;
  except
    Memol.Lines.Add('Error: ' + Exception(ExceptObject).Message);
  end;
end;

```



1.2.3 Non Delphi Client Examples

Enter topic text here.

1.2.3.1 Visual Basic

This Visual Basic example uses the [Microsoft SOAP Toolkit Version 3.0](#)

1.2.4 Asta SOAP Explorer

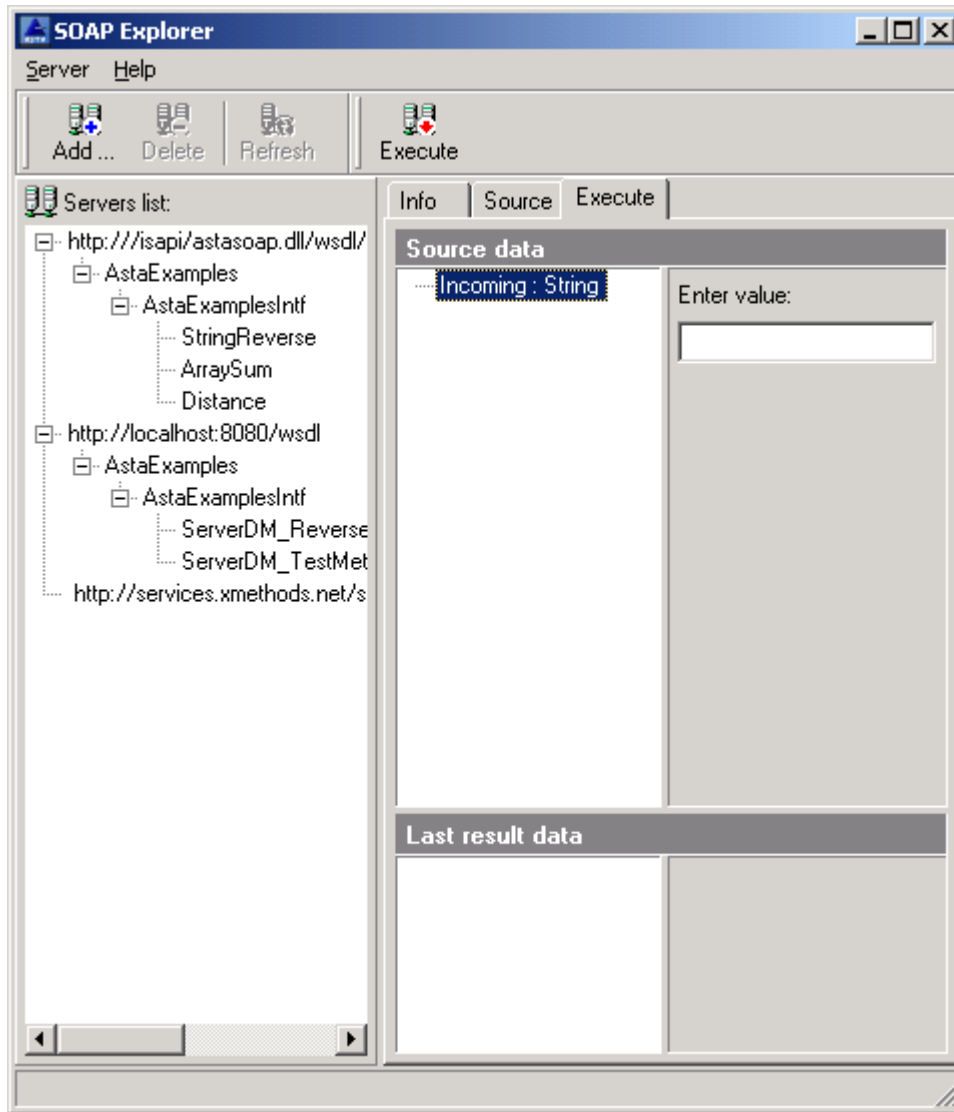
The AstaSOAP explorer comes in 2 versions: one using no third party VCL's and the other using EIPack from www.eldos.org. A current version of the AstaSoapExplorer is available for [download](#).

We've Tested the explorer against the following SOAP Services

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

<http://services.xmethods.net/soap/urn:xmethods-DomainChecker.wsdl>

<http://live.capescience.com/wsd/GlobalWeather.wsdl>
<http://reto.checkit.ch/Scripts/Lotto.dll/wsd/IgetNumbers>
<http://www.tankebolaget.se/scripts/NumToWords.dll/wsd/INumToWords>
<http://beta2.eraserver.net/webservices/mxchecker/mxchecker.asmx?WSDL>
<http://www.xmlbus.com:9010/xmlbus/container/Converter/ConverterService/ConverterPort>
<http://www.nickhodge.com/nhodge/finnwords/finnwords.wsdl>



1.3 AstaIO

1.4 ASTA 3 Soap Support

ASTA 3 allows for any Servermethod to be tagged as available for a soap service. There is an

example non-database ASTA server, from the ASTA help tutorials, in the
 \soap\examples\AstaServerBizobjectsSample directory.

there are 2 methods tagged so that they can be published as soap services

ServerDM_ReverseAction- Takes a string and returns the string reversed to the client

ServerDM_TestMethod - Takes an Integer Input and returns an output with 100
 subtracted from

it and also returns the server time as a time stamp

In normal ASTA clients, the notation ServerDM.TestMethod is used. The period cannot be
 used in

WebServers so we have substituted the underscore (_). future builds will allow for the
 Server

DataModule to be ignored so you can just call TestMethod or ReverseAction.

How it works (\Asta\Soap\Examples\SoapBizObjects)

=====

AstaSoapUtils.pas

TAstaSoapClientList- This class has a method of RegisterServer that allows for any server
 to

create SoapMethods that will map to any servermethods with the SoapService:Boolean
 set to true.

```
procedure RegisterServer(Address: string; Timeout: Integer; Port: Word);
```

Since this will be called from an http server or isapi dll a blocking call must be used so
 the

timeout value is available (in milliseconds) to be passed in to control how long the client

should block or wait in order to connect to the ASTA server to return a list of
 servermethods.

the TAstaSoapClientList will create a List of AstaClientDataSets that each have their own

AstaclientSocket configured to point to the server along with the servermethod name set for the

AstaclientDataSet.

You can register as many servers as you want. After you have registered servers, call

BindSoapMethods which will create a TAstaSoapmethod for each Servermethod found on the

registered servers.

Now when soap requests come in, the SoapMethod will invoke an AstaclientDataset/AstaclientSocket

to connect to an ASTA server and execute a servermethod, translating any params from the

AstaclientDataSet back to the soap method.

Todo: add encryption/compression/authentication/cookie options.
error handling if a server goes down.

have a call to get the list of available servers from a existing asta server
allow for non-servermethods to be called on Soap Servers.