# IndySoap Tutorial

Dave Nottage

Portions of this paper are borrowed from the IndySoap documentation.

## Contents:


- Foreword
  - Installing IndySoap
  - Creating a webservice with IndySoap
  - Defining server interfaces
  - Registering the interfaces and "SOAPable" classes
  - Generating Interface Type Information (ITI)
  - Creating the server
  - Deploying the server
- Creating a webservice client with IndySoap
  - Creating the client
  - Invoking the webservice methods
- Conclusion


## Foreword

This paper assumes the reader has a basic understanding of SOAP, WSDL, HTTP and XML.

## Installing IndySoap

### Installation:

The following procedure is recommended for installing IndySoap:

1. You need the latest Indy release (9.03 or more recent). You can obtain the latest source for Indy from:
   ftp://indy90:indy90@ftp.nevrona.com/

2. Create a SOAP Directory in your Indy directory

3. Copy the SOAP directory structure into the SOAP Directory.
   *To actually use IndySoap, you only need the files in the root soap directory, but the other files will be useful from time to time*

4. Add the <indy>/soap directory to your delphi library path

5. Review the VER140ENTERPRISE in IdSoapDefines.inc - you may want to enable it

6. [optional but recommended] Compile and run the DUnit tests in <indysoap>\dunit\IdSoapTests.dpr

7. Open the IdSoap.dpk file, then press the install button

### Deliverables Reference:

The following content is released as part of the IndySoap package:

<indysoap>Core Code for IndySoap, + design time package IdSoap.dpk\demoDemonstration programs with a brief tutorial\docoAssorted IndySoap documentation\dunitDUnit tests for IndySoap\loggerA utility for logging SOAP transport. This can be used to capture a SOAP conversion for submission to the IndySoap pit crew for investigation\managerPrograms to assist in the management of ITI generation\XPlatformCopies of programs used by the IndySoap pit crew to test interoperability issues

Important Points:

1. Refer below for a description of what the ITI is and why it is important

2. The type registry and the ITI must be equivalent on both client and server (both can contain information not in the other, but where the information overlaps, it must be identical

3. The Transport and Encoding layer choices are independent

4. The server implementations are fully multithread safe

5. The client side is thread safe, but a single client can only be used in one thread at a time

# Creating a webservice with IndySoap

### Defining server interfaces

The functionality available through SOAP is defined by the interfaces registered in the ITI (Interface Type Information - see below).

Parameters of complex types that are used in methods in IndySoap, need to be descended from TIdBaseSoapableClass (much like TRemotable in Delphi 6's SOAP implementation). Other parameter types are also supported; please refer to the IndySoap documentation for details. IndySoap also allows for documentation in the WSDL generated, which is obtained from special comments inserted in the interface definition. See the IndySoap documentation for further information on this subject.

A typical interface would be defined in a type declaration like this:

```
type
  TKeyInformation = class(TIdBaseSoapableClass)
  private
    FName : string;
    FStartKey : integer;
    FNextKey : integer;
  published
    property Name : string read FName write FName;
    property StartKey : integer read FstartKey
                                write FStartKey;
    property NextKey : integer read FnextKey
                               write FNextKey;
  end;

  TKeyList = array of TKeyInformation;

  IKeyServer = interface(IIdSoapInterface)
         ['{7C6A85A1-E623-4D6F-BA03-915BD49CE7D4}']
    function GetNextKey( AName : string ): integer;
         stdcall;
    procedure ResetKey( AUserName, APassword, AName :
         string; ANewValue : integer); stdcall;
    procedure ListKeys(out VList : TKeyList); stdcall;
  end;
```

There are a number of rules about the way interfaces can be defined for use with IndySoap

1. All Interfaces must descend from IIdSoapInterface which is defined in IdSoapTypeRegistry
2. All interfaces must have unique GUIDs (a certain IndySoap pit crew member always forgets to update them when copying and pasting interfaces!)
3. All types used as parameters must be registered with IdSoapTypeRegistry. See below for further information
4. All methods must be either functions or procedures
5. All methods must use the StdCall calling convention
6. Some simple parameter types are banned - see below for further information
7. Interfaces, pointers, methods, etc cannot be used as parameters
8. On the server, all interfaces must have a factory registered.

## Registering the interfaces and "SOAPable" classes

So that the IndySoap runtime can locate interfaces and "SOAPable" classes that you use as parameters, they need to be "registered" with IndySoap eg:

```
uses
  idSoapIntfRegistry;

initialization
  IdSoapRegisterType(TypeInfo(TKeyInformation));
  IdSoapRegisterType(TypeInfo(TKeyList), '', '',
         TypeInfo(TKeyInformation));
  IdSoapRegisterInterface(TypeInfo(IKeyServer));
```

## Generating Interface Type Information (ITI)

Refer to the IndySoap documentation for a detailed description of what ITI is, and why it came about.

ITI's are generated at compile time. The generation of ITI's is controlled by an ini file that generally has the file extension .IdSoapCfg. At present, the .IdSoapCfg file is generated manually, however the example file(s) from the demo could be used as a template. The following explains the sections of the file:

```
[Project]
;Not required
;Root dir for all files. Saves putting full path names on
all files
;If no dir is specified, the current directory is assumed
Dir=c:\indy\soap\demo

[Source]
;Required
;A list of files that contain interfaces to be parsed into
the ITI. At least one file must be listed
KeyServerInterface.pas

[Inclusions]
;Not required
;If any interface names are listed here, they will be the
only interfaces included in the ITI.
;The default is all interfaces for the Sources listed
IKeyServer

[Exclusions]
;Not required
;If any interface names are listed here, they will be
excluded from the ITI.
;The default is no exclusions
IKeyServer
```

```
[Output]
;Required
;Filename of the ITI output file.
BinOutput=KeyServer.iti

;Not required
;Filename where the XML copy of the ITI will be output, if
desired
XMLOutput=KeyServer.iti.xml
```

Note that sections, names and Filenames are case sensitive on Linux

There are a number of ways to actually make the ITI file generation happen:

- When a project is compiled by the IDE, and there is a .IdSoapCfg file with the same name as the project file, the ITI will be generated automatically.
- The IdSoapITIManager and related programs in <indysoap>\manager can generate ITI files interactively.
- If you run the IdSoapITIManager with the .IdSoapCfg filename as a parameter it will generate the ITI File.

### Creating the server

Start with a blank project, ie File|New Application (or File|New|Application for Delphi 6).

Use File|New|Other, Unit (or just File|New|Unit in Delphi 6), to create a unit for the interface, and save the file with a meaningful unit name. A convention is to include the name of the interface, suffixed with the word "Interface" in the unit name.

Create the interface as described in the "defining server interfaces" section. You can also refer to the KeyServerInterface.pas unit from the demo as a guide.

Create a **separate** unit for the implementation. This helps separate the client interface from the implementation. Save the file with a meaningful unit name. Again, a convention is to include the name of the interface, this time suffixed with the word "Implementation" in the unit name.

Include IdSoapIntfRegistry and the interface unit created in the previous step, in the uses clause, eg:

```
uses
  IdSoapIntfRegistry,
  KeyServerInterface;
```

For your implementation class, derive from TIdSoapBaseImplementation, the root class for IndySoap implementation classes, and implement the interface created in the previous step. Implement the methods for the class as you normally would for a class that implements an interface. Remember that the methods need to use the **stdcall** convention. Eg:

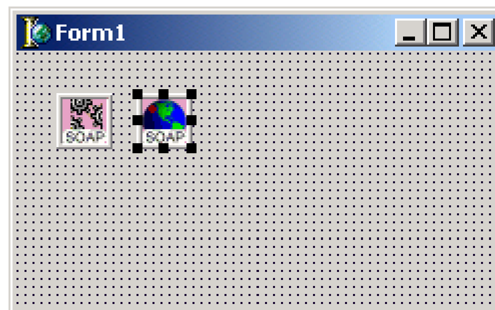
```
type
  TKeyServerImpl = class(TIdSoapBaseImplementation,
        IKeyServer)
    function GetNextKey ( AName : String ): integer;
             stdcall;
    procedure ResetKey ( AUserName, APassword, AName :
             string; ANewValue : integer); stdcall;
    procedure ListKeys(out VList : TKeyList); stdcall;
  end;
```

In the implementation section of the unit, register the interface class:

```
initialization
  IdSoapRegisterInterfaceClass('IKeyServer',
    TypeInfo(TKeyServerImpl), TKeyServerImpl);
```

Refer to the KeyServerImplementation.pas unit from the demo as a guide.

Drop a TIdSoapServer, and TIdSoapServerHTTP onto the main form:




… and link the two by setting the SOAPServer property of the TIdSoapServerHTTP to the TIdSoapServer. Set Active to True on the TIdSoapServerHTTP. Note: you can add additional transports (eg TCP) by simply adding the corresponding component (eg TIdSoapServerTCP) to the form, and linking it to the TIdSoapServer. Support for additional transport types is discussed in the IndySoap documentation.

Set the ITISource property on the TIdSoapServer to islFile so that the ITI is retrieved from the .iti file at runtime, and use either the ITIFileName property or the OnGetITIFileName event to provide the filename. [Figure 1]

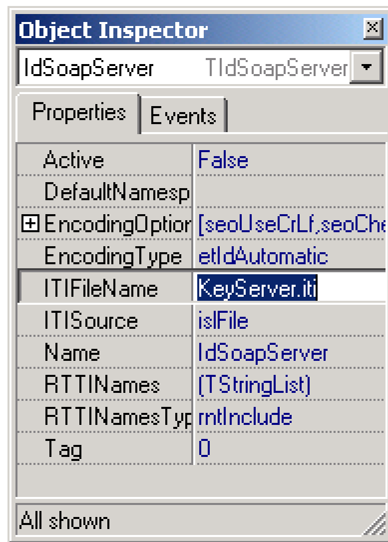



Figure 1

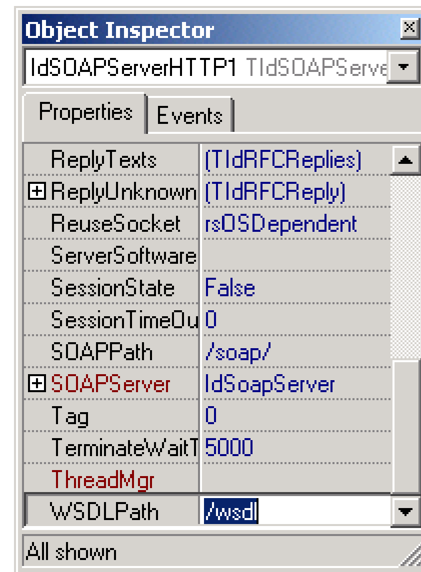



Figure 2

Refer to the IndySoap documentation for other ITISource options, and for the various encoding options available.

Once the ITISource and associated properties have been set, set the Active property to True. Note: due to a minor bug in the beta, the Active property needs to be set to True at run-time, so code the form's OnCreate to do this:

```
procedure TForm1.FormCreate(Sender: TObject);
begin
  IdSoapServer1.Active := True;
end;
```

You can leave the SOAPPath and WSDLPath properties on the TIdSOAPServerHTTP as they are, however it is worth taking note of them for future reference, especially when building the client. The defaults are shown in the above image: [Figure 2]

You can change the port on which the server operates on by setting the DefaultPort property. In the demo, it is set to 2445.

This completes the server.

### Deploying the server

Compile and deploy to whichever directory you choose, making sure that if an .iti file is used, that it is deployed to the directory indicated by the ITIFileName property, or the path used in the OnGetITIFileName event.

You can browse the resulting WSDL by pointing a web browser at the server WSDL path, eg: http://localhost:2445/wsdl/

# Creating a webservice client with IndySoap

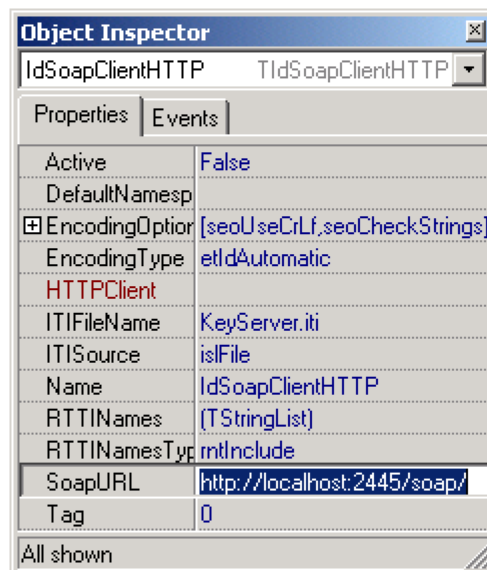
## Creating the client

For the client, you will need the .ITI file from when the server was created, or create the .ITI file using just the interface source file. In the future, .ITI files will be created from WSDL published by the server.

Start with a blank project, ie File|New Application (or File|New|Application for Delphi 6).

Drop a TIdSoapClientHTTP onto the form (or TIdSoapClientTCPIP, depending on whether the server supports it):

Set the ITISource and ITIFileName properties as per the instructions for the server. (See Creating a webservice with IndySoap). Set the SoapURL property to the SOAPPath described by the server, eg: http://localhost:2445/soap/




i.e. in this case, the webservice is running on the same machine (localhost) on port 2445, and the path to the service is soap/ (as per the server example)

Include the interface unit in the implementation uses clause, eg:

```
uses
  KeyServerInterface;
```

### Invoking the webservice methods

To call a method of the webservice, cast the TIdSoapClientHTTP instance to the interface, and invoke the method eg:

```
procedure TForm1.Button1Click(Sender: TObject);
var
   IKey : IKeyServer;
begin
   IKey := IdSoapClientTCPIP1 as IKeyServer;
   eKeyValue.Text :=
      inttostr(IKey.GetNextKey(eKeyName.Text));
end;
```

You can refer to the demo client (KeyClient) for an example.

### Exception Handling

Exceptions raised on the server are passed to the client as a SOAP fault, and are re-raised in the client.

## Conclusion

Creating webservices with IndySoap is easy. It will be even easier in future versions, which will include the ability to create ITI from WSDL, and possibly seamless integration of ITI into the executable.

IndySoap has been very carefully designed to be robust and extensible, as described in Grahame Grieve's paper that accompanies this paper.

---

Dave Nottage is CTO of Pure Software Technology, and is an independent software consultant, specialising in Delphi and Kylix. He can be contacted at:
davidn@smartchat.net.au