ASTA 3 for Delphi

ASTA SkyWire

Wireless Enabled Web Services and Collaborative Tools

---

by ASTA Technology Group

ASTA Skywire supports cross platform clients and server to allow for developers to create fast, reliable and secure applications that run over any network.

Table of Contents

ASTA Hitchhiker's guide to the wireless universe

Run over any TCP/IP connection and be able to disguise yourself as http at anytime.

Have a middleware server that supports ASP and Peer to Peer.

Be aware that the most precious resource is bandwidth.

Provide XML Support.

Provide Compression Options out of respect for #2.

Support Web Services on both client and servers.

Provide Encryption Options to be able to operate securely.

There shall be no administration required (Zero Admin rule).

Authenticate at the server by username and password and be able to uniquely identify remote devices.

Once a client app is deployed it should be able to update itself automatically and efficiently.

Be able to work connected and disconnected with sync technology to bridge those states.

Work with the most popular Development tools.

Work with any Database, on any Platform and provide a way to move data between any database and any device.

Have a component architecture so that the same techniques can be used cross platform

1 ASTA 3

The Internet Second Wave is here and the ASTA Technology Framework provides the foundation to not only build scalable rich, thin client applications but to add collaborative tools and full cross platform support beyond the Desktop.

Most ASTA developers are Database Application Developers and start with a database centric view of the world. ASTA wants to remind you that "There is more to Heaven and Earth Horatio, than Win32 Database applications"

ASTA allows you to build cross platform, thin client, zero admin, secure, scalable Database applications that can run over the Internet. We want you to concentrate on coding your application rather than dealing with low level issues like threading or writing SQL update statements.

Below is a list of general Topics covering ASTA that you can refer to, in order for you to see what is possible. And ASTA is committed to making what is possible, easy to implement.

Database	In Memory DataSets Client Side SQL vs coding the Server Server Side Techniques Updating the Database Refetching Data Broadcast Changes
Deployment	AutoUpdate
Application Design	Code the Client Code the Server
Messaging	Sending a String Sending a Stream Send Anything: TAstaParamLists Instant Messaging API (IMA)
State	Clients with State TCP/IP
Stateless Clients	HTTP and WinInet Cookies Blocking Calls
Firewalls	Firewalls
Security	Encryption Logins
Scalability	Scalability
Servers	Service or EXE Remote Control Remote Admin Scalability Database Considerations
XML	XML
SOAP	SOAP
File Transfer Routines	File Transfer Routines

1.1 ASTA 3 New Features

In Memory DataSets

Index Support	All ASTA datasets now have indexes for findkey and set range support and optimized locate calls to use indexes where available
Extended Contraints	We've extended constraint support for all ASTA DataSets
New TAstaCloneDataSet	Allows for design time or run time copying of any TDataSet to an ASTA In Memory DataSet
Advanced Streaming Options	Adds compression and encryption options to SaveToFile calls
Extended XML Support	ADO and Midas formats supported using the native ASTA XML parser or the Microsoft Parser using a "thunk" layer built into ASTA 3
Filters Optimized	ASTA filters were completely re-written with a smaller footprint and less depenencies on Borland code
Aggregates	DataSets support Aggregates like Sum and Avg

SQL Generation

Params for SQL Generation	Params are now used in all SQL generation deprecating the requirement to set AstaClientSocket properties like DateTimeMaskforSQL
New TAstaUpdateSQL Component	Allows for the SQL that ASTA generates to be supplemented
	Indexes are used internally for bookmarks to increase the speed of SQL generation on large datasets. Bulk inserts are also significantly faster.
	NoSQLFields property on AstaClientDataSets allows for columns to be tagged so they don't generate SQL. In the past columns had to be tagged as readonly.

Security

	Strong Encryption options currently available to the US and Canada only. We have submitted the papework to allow for distribution outside of North America.
AstaDES	Built in DES Encryption with design time support
AstaAES	Built in AES (Rijndael) Encryption (US and Canada only)
AstaRSA	ASTA RSA implements RSA Public Key Exchange Encryption. Available on Palm and WinCE also. (US and Canada only)
	Added the concept of Trusted Address

XML

ASTA Native XML Parser	Pascal DOM XML parser or thunk layer using the Microsoft Parser or Delphi 6 components. Thunk layer available in source code only licenses.
ASTA XML Explorer	XML Explorer shows how to use the ASTA DOM Parser
XML Thunk Layer	Allows the ASTA XML parser to be used or the Microsoft Parser

HTTP Tunneling

ASTA clients can tunnel through any firewall using ASTA Sockets or WinInet. Use of winInet is now uses dynamic loading of WinInet.dll
Enhanced support for Cookies and Logins via stateless HTTP
Stateless Messaging allows for Instant Messaging via Http or disconnected users

AutoUpdate

The autoupdate process has been extended to allow for the Updated EXE to be streamed from http servers and to also integrate the ASTA Binary Patch Manager to allow for just "delta's" or binary patches to be streamed down and applied by remote clients.

ASTA Servers

	ASTA servers are available in a new format that allows for them to be run as normal EXE's or NT Services. Additionally they can be easily extended to support the following technologies (typically with just a property setting): Remote Control Instant Messaging PDA support ODBC Driver JDBC Driver
New Threading Model	New tmSmartThreading allows for ASTA Pooled and Persistent Sessions threading to be mixed allowing for the best of both worlds.Scalable servers using connection pooling but also for any user being able to connected using a specific user name and password.
	Remote Admin built into the AstaServerSocket to change server settings remotely.

ODBC Driver

Servers support the ASTA thin client ODBC driver available using the ASTA Safari Kit

JDBC Driver

Servers support the ASTA type 3 JDBC driver available using the ASTA Safari Kit

PDA Support Extended

New TAstaPdaPlugin allows for SkyWire PDA support with one line of code

C# Client

Dot net support available from the ASTA safari kit

COM Client

Non VCL Com client available from the safari pack using Wince of Win32

Messaging Options

Extended Messaging options using new OnOutParamList event on the AstaServerSocket and threaded Database enabled blocking messaging calls.
New SendGetCodedDBParamList calls allow for safe threading database access in ParamList messaging
Options to send and get large files from servers to clients and clients to servers in configurable "chunks"

Instant Messaging

Formal Instant Messaging API (IMA) to support sophisticated IM applications including disconected messaging with Java, Palm and WinCE IM clients available.
Instant Messaging Events on the AstaClientSocket and Instant Messaging Options on ASTA server Socket.
Disconnected Messaging supported with the New TAstaMQManager
Named Messaging support with SendNamedCodedParamList

Remote Control

ASTA Remote Control source was given to ASTA 2.6 users as a XMAS present in Dec 2001. It is now easily integrated into ASTA 3 servers with a line or 2 of code and allows for pcAnywhere type of access to the remote Server. ASTA licensing has been relaxed to allow for RemoteControl Servers to be run from client apps so that ASTA users can support their clients by connecting directly to their machines and "taking control" of the application from a remote admin client.

XML Based Code Manager

ASTA XML based tutorial manager allows for any source files to be parsed and comments to be added and saved and displayed using XSLT applied to XML that the manager generates. Will be used to document ASTA server and tutorials.

Provider Broadcasts

New manual provider broadcasts	Additional properties have been added to make broadcasting and updating possible with no code.

Cached MetaCalls

When setting servermethods or provider names at runtime, ASTA clients must fetch param information. This is optionally cached locally to avoid a trip to the server.

Enhanced Logins

Logins now include a ClientParamList returned to clients on logins and also allows for the pickup and any messages sent to other users while off line.

Design time Security Options

Trusted Addresses can now be defined on ASTA servers to allow for design time access to named IPAddresses only.

Multiple Network Card Support

Servers can be bound to specific network cards.

KeepAliveOptions

Some networks disconnect after certain periods of activity. The AstaclientSocket has a new KeepAlivePing option that sends a codedMessage to ASTA servers to keep dialup lines active.

Voip Support

ASTA Voice over IP is in Beta. This is an ASTA addon.	Is not part of ASTA 3 but will be integrated with ASTA 3 instant messaging and TAstaMQManager to provide for administrative features for VoIP apps.

ASTA Scheduler

We have licensed the Scheduler component written by David Quinn from ASTA UK and this will be integrated in ASTA 3 to work with the TAstaMQManager to allow for events to be scheduled and exececuted in threads.
ASTA Scheduler probably won't make it into the initial release of ASTA 3 but will come soon after.

ServerMethods integrated with PDA Support

ServerMethods integrated with PDA support
ServerMethods can now be called from Palm and WinCE clients directly

1.2 ASTA 2.6 to ASTA 3 Migration Issues

ASTA 2.6 and ASTA 3 are largely compatible as we have taken great care in making sure that your existing applications will migrate easily. This document will discuss migration issues from ASTA 2.6 to ASTA 3.

1. SQL Generation now uses Params
In ASTA 2.6 SQL was expanded so that Dates and Floats would have to be properly formatted depending on the settings of the machine running the AstaServer. In ASTA 3, all SQL generation uses Params. In order to do this, there are a couple of new events on the AstaServerSocket in order to support this:

OnSubmitSQLParams
OnServerQuerySetParams

All of the properties used on the AstaSQLGenerator and AstaClientSocket to format SQL have become deprecated.

2. ASTA Servers are now in a new format to run as normal exe's or services. There is a new AstaServiceUtils unit that encapsulates all that is necessary to run servers as services. ASTA 3 Server's main forms descend from the TAstaServceForm in order to run as services or normal exe's. In addition, there are built in concepts of Remote Control and Remote Administration.

3. Depending on the threading model (OnClientLogin, OnClientDBLogin). ASTA 3 adds a new event, OnClientAuthenticate that handles all threading models and also handles logins from stateless HTTP clients with the new "Cookie" support. The OnClientAuthenticate event has an extra ClientParams:TAstaParamLists parameter that returns to the client in the new OnLoginParamsEvent. This allows for additional information to be streamed back to clients within the login process.

Migrate to OnClientAuthenticate on the server
Migrate to OnLoginParams on the client

4. ASTA server add a new threading model that implements Pooled sessions but also allows for any remote client to run their own PeristentSession and to switch back and forth from Pooled or Persistent. It is recommended that all servers run tmSmartThreading.

5. Indexes are now available for AstaDataSets. All AstaDataSets and AstaClientDataSets now have Indexes. In the past, the way to re-order datasets was to sort them. The old sort calls still work, but create temp indexes. Calls like SetRange are now supported. You probably want to rethink calls like SortDataSetByFieldName and SortOrderSort to use the new indexes.

6. XML- ASTA 2 DataSets supported XML and ASTA 3 still supports the old ASTA 2 XML routines. But ASTA 3 has a new DOM XML Parser and adds supports for standard ADO and DataSnap formats as well as AstaParamList XML Routines so it is recommended that the new XML DOM support be used.

1.3 ASTA History

ASTA has been in development since 1997 after Delphi Chief Architect Chuck Jazdzewski visited the Boise Delphi User Group and told us about the new open TDataSet model in Delphi 3 along with the new borland sockets. Within 2 weeks we had AstaClientDataSets streaming across the Internet.

It took us 7 months then to develop ASTA 1.0 which was released in July of 1998 after a large beta. ASTA 2 was released in October 1999 adding server side components like the TAstaProvider and TAstaBusinessObjectsManager. ASTA Palm and Wince support was added in the year 2000 and AstaInterOP for Windows and Linux for cross platform support was released in 2001.

Our design goal in building ASTA was to:

1. Allow Database Developers to use their existing skills to develop N tier Applications so that existing applications could be quickly ported to ASTA using Client Side SQL
2. Allow Experienced N Tier developers to use their N Tier Skills in extending ASTA servers using advanced N Tier features
3. Build very scalable and easy to deploy servers and handle all Threading Issues internally so that Developers could concentrate on building applications
4. Allow Thin Client Applications that could be deployed with NO Dll's and build in the ability to have client exe's update themselves when new versions were registered on the server
5. Require only SQL Select statements so that all insert, update and delete SQL could be generated by ASTA components
6. Build an easy to use Messaging Layer so that Database Application Developers that had NO tcp/ip experience could easily stream any kind of data across the internet
7. Abstract the server side so that any Delphi 3rd Party Component could be easily plugged into an ASTA Server

To that we have since added the ability to go cross platform and with any device with the ASTA SkyWire Architecture. Our design goals have evolved to our Hitchhikers Guide to the Wireless World.

The ASTA train is on track and moving forward to bring you the latest technology and making it easy for you to use. The ASTA team is commited to doing whatever it takes to assure that YOUR ASTA project is successful!

Steve Garland, CEO
ASTA Technology Group Inc
Boise Idaho USA
February 2002

1.4 ASTA 3 Tutorials

ASTA 3 ships with many tutorials. Some include their own servers, others use standard ASTA servers to run.

You can view the tutorials in several groups.

AstaDataSets	In memory Datasets
AstaClientDataSets	In Memory DataSets that communciate with a remote server. These tutorials require an ASTA Server to be running
Messaging	ParamList Messaging and File Transfer Examples
Server Side Techniques	Shows how to use TAstaProviders and ASTA Server Methods using the TAstaBusiness Objects Manager to implement "Business Logic" on the Middle Tier
Feature Tests	Shows some examples of how to take inventory of ASTA server components from remote clients with ServerMethod Testers, Provider Testers, SQL Demo clients and the AstaSQLExplorer
Security Issues	

1.4.1 DataSets- In Memory

TAstaDataSets

Create Fields at runtime
Indexes
Master Detail

1.4.1.1 Master Detail

Shows how to use masterDetail with an in memoryDataset and to flip back and forth between master/detail support with setting and clearing the master fields property.

AstaClientDataSets typically implement master/detail using a Parameterized Query on the Detail DataSet. As the master rows are changed new queries are sent to the server. ASTA also provides a way to query all the detail for all the master rows and to use a local filter to implement master detail. There are helper calls to toggle back and forth between "connected" master/detail and disconnected Master/Detail.

```
procedure TForm1.ButtonClick(Sender: TObject);
begin
with DetailDataSet do
if Button.tag=0 then begin
 SetToDisconnectedMasterDetail('Select * from orders order by orderno','CustNo');
 Button.Tag:=1;
 Button.Caption:='Set to run Master Detail from Server';
end else begin
 SetToConnectedMasterDetail('SELECT * From ORDERS  where Custno=:CustomerNumber');
 Button.Caption:='Set to run Master Detail Locally';
 Button.Tag:=0;
end;
end;
```

1.4.1.2 Indexes

TAstaDataSets support indexes so you can use the standard Delphi TTable calls like SetRange and FindKey. Additionally, any calls to Locate will use an existing Index so that lookups are optimized.

The Index Tutorial shows examples of using TAstaDataSet index methods.

if an index is addedwith no fieldname it will index on the bookmark. To addindexes with code shortcuts use

```
   procedure AddIndexFields(TheIndexName:String;Const FieldNames:Array of String;Const
Descending:Array of Boolean);
  procedure AddIndex(FieldName:String;Descending:Boolean);
```

Below shows code to add indexes the traditional way.

```
AstaDataSet1.Active := True;
  Listbox1.items.Add('<not sorted>');
  with AstaDataSet1 do begin
    Indexes.Clear;
    with Indexes.Add do begin
      Name := 'Country';
      Fields:='Country';
      Active := True;
      Listbox1.items.Add(Name);
    end;
    with Indexes.Add do begin
      Name := 'Branch';
      Fields:='branch';
      Active := True;
      Listbox1.items.Add(Name);
    end;
    with Indexes.Add do begin
      Name := 'Sequence';
      Fields := 'Sequence';
      Options := [ioUnique];
      Active := True;
      Listbox1.items.Add(Name);
    end;
    with Indexes.Add do begin
      Name := 'Transaction_Id';
      Fields := 'Transaction_Id';
      Active := True;
      Listbox1.items.Add(Name);
    end;
  end;

end;
```

1.4.1.3 Create Fields

Description

AstaDataSets are in memory Datasets that can be configured to use any Delphi type of DataType by using the FieldsDefine property at design time or at runtime. CreateFields shows how to use the FastFieldDefine call to add fields and also to add Calculated Fields at runtime.

FastFieldsDefine

To add fields at runtime use FastFieldsDefine. If the Dataset was created and fields are being added the first time, you do not have to call NukeAllFieldInfo but if you are re-suing the DataSet call NukeAllFieldInfo to clear out all field information.

```
with AstaDataSet1 do
  begin
    close;
    nukeallFieldInfo; //only needed if you are going to recreate it again
    FastFieldDefine('Name', ftstring, 25);
    FastFieldDefine(FormatDatetime('tt', now), ftinteger, 0);
    FastFieldDefine('Date', ftdatetime, 0);
    FastFieldDefine('Age', ftinteger, 0);
    Open;
  end;
```

1.4.1.3.1 Add a Calculated Field at runtime

Description

When you right mouse from design time in Delphi on a Dataset you are presented with the fields editor and can create PersistentFields at design time. when you want to do the same at runtime, you need to make the fields persisent. Below is an example of using the MimicTableandMakeFieldsPersistent method from AstaFieldandStream.pas to accomplish this.

```
Var
  CalcField: TStringField;
begin
  with AstaDataSet1 do begin
    close;
    NukeAllFieldInfo;
    with FieldDefs do
    begin
      Add('Test', ftString, 25, false);
      Add('Name', ftstring, 25, True);
      Add('Date', ftDateTime, 0, False);
      Add('Age', ftinteger, 0, False);
      Open;
    end;
    if not CalcFieldCheckBox.Checked then exit;
    //this next call is like right mouse at design time and adding fields
    MimicTableAndMakeFieldsPersistant(AstaDataSet1, False);
    CalcField := TStringField.Create(self);
    CalcField.FieldKind := fkCalculated;
    CalcField.Size := 20;
    CalcField.FieldName := 'CALC';
    CalcField.DataSet := AstaDataSet1;
    AstaDataSet1.Open;
  end;
end;
```

1.4.2 Client Side SQL

AstaClientdatasets descend from TAstaDataSets and are in memory DataSets that communicate with remote servers via the TAstaClientSocket.These tutorials show how to use the various methods and properties of the AstaClientDataSet.

Aggregates
Cached Updates
Constraints
Indexes
Master Detail
Sorting
SuitCase
Transactions

1.4.2.1 Aggregates

Requires

AstaBDEServer to be running

Description
Use Aggregates to define aggregates that summarize the data in the client dataset. Aggregates is a collection of TAstaAggregate objects, each of which defines a formula for calculating an aggregate value from a group of records in the client dataset. The individual aggregates can summarize all the records in the client dataset or subgroups of records that have the same value on a set of fields. Aggregates that summarize a subgroup of records are associated with indexes, and can only be used when the associated index is current.

This example shows how to define an Aggregate called GrpCnt.

```
procedure TForm1.FormCreate(Sender: TObject);
begin
  with AstaClientDataSet1 do begin
    Indexes.Clear;
    with Indexes.Add do begin
      Name := 'BMK';
      Active := True;
    end;
    with Indexes.Add do begin
      Name := 'Company';
      Fields := 'Company';
      Options := [ioDescending, ioUnique, ioNatural];
      Active := True;
    end;
    with Indexes.Add do begin
      Name := 'TaxRate';
      Fields := 'TaxRate';
      Active := True;
    end;
    with Indexes.Add do begin
      Name := 'CustNo';
      Fields := 'CustNo';
      Options := [ioUnique];
      Active := True;
    end;
    with Indexes.Add do begin
      Name := 'State';
      Fields := 'State';
      Options := [];
      Active := True;
    end;
  end;
  AstaClientDataSet1.IndexName := 'State';
  with AstaClientDataSet1 do begin
    Aggregates.Clear;
    with Aggregates.Add do begin
      Name := 'GrpCnt';
      AggKind := akCount;
      IndexName := 'State';
      Level := 1;
      Active := True;
    end;
  end;
  LoadIndexes;
  ListBox1.ItemIndex := 1;
  ListBox1Click(nil);
  LoadAggs;
  ListBox2.ItemIndex := 0;
  ListBox2Click(nil);
end;
```

1.4.2.2 CachedUpdates

Requires

AstaBDEServer to be running

Description
ASTA was designed so that you only need write Select SQL and not have to write SQL for inserts, updates or deletes as ASTA will generate SQL for you. This way you only have to worry about populating DataSets by using either ClientSide SQL, TAstaProviders or ServerMethods with the TAstaBusinessObjectsManager.

Cached Updates are the recommend way to post changes to remote ASTA servers. When you run this demo, just edit a couple of rows and you will see the OldValuesDataSet that the AstaClientDataSet creates internally. By tracking changes, the AstaClientDataSet can support RevertRecord and CancelUpdates.

In normal client server programming, to post changes to a server you start a transaction, ExecSQL and then Commit or Rollback as appropriate. When AstaClientDataSets you need only set the EditMode to Cached and then call ApplyUpdates(usmServerTransation) and ASTA will generate all the SQL for you, go to the server, Start a transction and Commit or Rollback as appropriate. All in one server round trip.

In order to see the actual AstaClientDataSet.OldValuesDataSet, you merely need to set up a grid and DataSource and set it at runtime in the AstaClientDataSet.AfterPost event.

1.4.2.3 Constraints

Requires
AstaBDEServer to be running

Description
Using ASTA Expressions you can define:

Custom/Imported constraints for each TField (field level)
These constraints will fire each time, when a new value is assigned to a TField. If a constraint is violated, an exception will be raised and no update to the TField will happen.

Custom/Imported constraints for TAstaDataSet
Constraints (record level). These constraints will fire each time, when a TDataSet.Post will occur. If the constraint is viloated, then no Post will happen.

DefaultExpression for each TField
This expression will be evaluated when a new record is created (Insert / Append) and the result will be assigned to TField.

Calculated field using DefaultExpression
The TField value will be calculated using this expression.

Expression language is the same as it is described in Delphi Help topic TClientDataSet.Filter

This demo works with DBDEMOS:Orders table. There are:
- a few record level constraints
- a few field level constraints
- a calculated by expression field AmountNotPaid

1.4.2.4 Indexes

Requires

AstaBDEServer to be running

Description

AstaClientDataSets support Indexes and Aggregates as they descend from the full featured TAstaDataSet This tutorial shows how to use indexes with the TAstaClientDataSet that is the same way it use used with the TAstaDataset example.

1.4.2.5 MasterDetail

Requires

AstaBDEServer to be running

Description

In order to use Master/detail with AstaClientDataSets the Detail DataSet needs to use a Parameterized query. After setting the Parameter DataType you then need to use the MasterFields/MasterSource property editor to hook to the master DataSet.

At runtime, a as the master row changes, a new paramterized query is sent to the server by the detail dataset. ASTA also supports disconnected Master/Detail which allows for a filter to be used by the Detail DataSet.

There may be times where you do not want to go to the server for every master row change but want to support master detail locally. This requires the SQL on the detail dataset to fetch all the rows for all the masters in the master Query. Caution is urged when using this technique in not bring to many rows over the wire.

1.4.2.6 Sorting

Requires

AstaBDEServer to be running

Description

AstaDatasets support indexes but there maybe times when you don't want to define indexes and just do quick sorts. This demo shows how to use SortDataSetbyFieldName and SortDataSetbyFieldNames.

```
procedure TForm1.GridTitleClick(Column: TColumn);
begin
 AstaclientDataSet1.SortDataSetByFieldName(Column.FieldName, RadioGroup1.ItemIndex=1);
(******************
 To sort multiple fields use
   procedure SortDataSetByFieldNames(const FieldNames: array of string; const Descending:
array of Boolean);
(*******************)
```

```
end;

procedure TForm1.Button1Click(Sender: TObject);
begin
 AstaclientDataSet1.SortDataSetByFieldNames(['ContactTitle','ContactName'],[false,True]);

end;
```

1.4.2.7 SuitCase

Requires

AstaBDEServer to be running

Description

ASTA supports a disconnected user or "suitCase" model. This tutorial shows you how to connect to a server, fetch data, save it to a file, empty the dataset, load it from a file, do edits and connect back to a server to post updates.

1.4.2.8 Transactions

Requires

AstaBDEServer to be running

Description

Shows a simple example of calling ApplyUpdates with EditMode set to Cached. Use the EditMode property editor to configure the UpdateTablename and Primekey fields.

1.4.3 Messaging

ASTA has a very flexible and easy to use messaging layer. For Database Application Developers, using Messaging to add features to Database applications opens up a world of possibilities as server "push" and peer to peer messaging can add many features never before deemed possible or so easy to business applications.

ASTA ParamList messaging is the recommended way to transfer any kind of data and AstaParamLists are available across just about all platforms with Java and C++ libraries for Win32, Linux, Palm and WinCE. To send very large files between client and server use the ASTA File Transfer routines that allow for large streams to be sent in configurable "chunks"

ParamList Messaging
ParamList Messaging with Threaded Database Support
Packing Up Data
File Transfer Techniques

Messaging examples come with both servers and clients and the servers are not database servers. To create an AstaServer to test or use with messaging and no database threading, you just need to make sure you have coded and assigned an Application.OnException event so that exceptions are not raised in dialogs on the server.

Below is an example.

```
procedure TForm1.AstaException(Sender: TObject; E: Exception);
begin
  //log the message on production servers either don't log to a UI control
 //or clear it every now and again!
  memo1.lines.add(E.Message);
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  //all ASTA servers need to supply an Application Exception Handler
  application.OnException := AstaException;
  astaserversocket1.active := True;
end;
```

1.4.3.1 CodedParamList

This tutorial includes a client and a server.

Server Side

The server is coded to handle 2 events:OnCodedParamlist and OnOutCodedParamList.

```
procedure TForm1.AstaServerSocket1OutCodedParamList(Sender: TObject;
  ClientSocket: TCustomWinSocket; MsgID: Integer; InParams,
  OutParams: TAstaParamList);
begin
/// Just need to fill up the OutParams for anything going back to the client
memo1.lines.add('incoming from SendGetCodedParamList:'+InParams.AsDisplayText);
OutParams.fastadd('Server','Server running at
'+GetThePcsIPAddress+':'+IntToStr(AstaServerSocket1.Port));

end;

procedure TForm1.AstaServerSocket1CodedParamList(Sender: TObject;
  ClientSocket: TCustomWinSocket; MsgID: Integer; Params: TAstaParamList);
begin
 memo1.lines.add('incoming from SendCodedParamList:'+Params.AsDisplayText);
end;
```

Client Side

This client example shows how to use SendCodedParamList and SendGetCodedParamList. It loads up a text file into a memo and also sends it as a stream, just for example purposes to show how flexible the AstaParamLists are.

```
procedure TForm1.BitBtn1Click(Sender: TObject);
var
  Params, RetParams: TAstaParamList;
begin
  if not OpenDialog1.Execute then Exit;
  Memo1.Lines.LoadFromFile(OpenDialog1.FileName);
  Params := TastaParamList.Create;
  try
    Params.FastAdd(OpenDialog1.FileName + ' File Size of ' + IntToStr(Length(memo1.Text)));
    Params.FastAdd(Now); // accepts TDateTime but Params[1] can be used as Params[1].AsString
or Params[1].AsDateTime
    Params.FastAdd(Memo1.Text); // text is a string representation of a TStringList
    Params.FastAdd(OpenDialog1.FileName,''); //here we are adding it as a stream
    //use any way you want
    params[params.count-1].LoadFromFile(OpenDialog1.fileName);
    if SendGetCheckBox.Checked then begin
      try
        RetParams := AstaClientSocket1.SendGetCodedParamList(15, Params);
      //SendGetCodedParams sends the Params to the Server and waits for a return result from
the
      //server. The Server must coded the OnOutCodedParamList and fill the OutParams
      //which gets received as the Function Result of SendGetCodedParamList
        if (RetParams <> nil) and (RetParams.Count > 0) then
          showmessage('Returned From the server server ' + #13 + RetParams[0].AsString);
      finally
        RetParams.free;
      end
    end else AstaClientSocket1.SendCodedParamList(15, Params);
  finally
    Params.Free;
  end;
end;
```

1.4.3.2 CodedDBParamList

Requires

Asta3ADOServer to be running since this is a Database Messaging routine.

Description
ASTA messaging has been used in more and more ways over the years and ASTA 3 introduces a fast and easy way to use ASTA messaging with the new call of SendGetCodedDBParamList. This allows you to use Asta message calls that will check out a database session on the server and execute your server side code in a thread.

Client side
=======

```
procedure TForm1.Button1Click(Sender: TObject);
var
pout,PIn:TAstaParamList;
i:Integer;
begin
 pout:=TAstaParamList.create;
 pin:=cs.SendGetCodedDBParamList(1000,POut);
 try
 for i:=0 to pin.count-1 do
  memo1.lines.add(pin[i].Name+':'+pin[i].AsString);
finally
 pout.free;
 pin.free;
end;
end;
```

Server side
=======
This new event is available which will be called in a thread with a datamodule handed off to you with any and all of your database components. You can also create and destroy DB components on the fly.

Just Grab the incoming Params and fill up the OutParams and they will be transported back to the client in a thread. if an exception occurs it will be raised at the client. or you can raise it yourself on the server to be transported back to the client.

```
procedure TServerDM.ServerSocketOutCodedDBParamList(Sender: TObject;
  ClientSocket: TCustomWinSocket; DBDataModule: TComponent; MsgID: Integer;
  InParams, OutParams: TAstaParamList);
begin
with dbDataModule as TAstaDataModule do begin
 query.sql.text:='select * from customers';
 query.Open;
 OutParams.FastAdd('Count',Query.Recordcount);
end;
end;
```

1.4.3.3 PackingUpData

ASTA Paramlist messaging provides an easy to use and efficient transport to send any type of information over any network. Sometimes, you may want to transport some complicated data. These tutorials show some techniques that may help.

DataSets Packed in ParamList
DataSet Packup

1.4.3.3.1 DataSetParamList

Requires

This demo uses the BDE but you can substitute any TDataset/database component.

Description
This demo shows how to add DataSets to AstaparamLists.

The following code adds the SQL as defined on the form as the ParamName and then copies the Data from a TQuery by using the AsDataSet Call of the TAstaParamItem.

```
procedure TForm1.BitBtn1Click(Sender: TObject);
var
  i: Integer;
begin
  FparamList.Clear;
  ListBox1.Items.Clear;
  for i := 0 to memo1.lines.count - 1 do begin
    Query1.Close;
    Query1.Sql.text := 'Select * from ' + memo1.lines[i];
    Query1.Open;
    ListBox1.items.add('Select * from ' + memo1.lines[i]);
    FParamList.FastAdd('Select * from ' + memo1.lines[i]);
    FParamList[FParamList.Count - 1].AsDataSet := Query1
  end;
  ExtractButton.Visible := FParamList.Count > 0;
end;
```

This code uses the TAstaParamItem.AsDataSet property to get the DataSet from any ParamItem.

NOTE : You are responsible for freeing any DataSet returned from the AstaParamItem.AsDataSet call as a TAstaDataSet is actually created in this call.

```
procedure TForm1.ExtractButtonClick(Sender: TObject);
begin
  if ListBox1.ItemIndex < 0 then raise Exception.Create('highlight a query!');
  DataSource2.DataSet.Free;
  DataSource2.DataSet := FParamList[ListBox1.ItemIndex].AsDataSet;
end;
```

1.4.3.3.2 DataSetPackup

Requires

This demo uses the bde but you can substitute any TDataset/database component.

Description
This demo shows how to clone a dataSet to an AstaDataSet (In Memory) and then store it in a blob or memo field of another DataSet and then to extract it.

The two calls used are

```
function CloneDataSetToString(D: TDataSet): string;
function StringToDataSet(const S: string): TAstaDataSet;
```

This code shows the packup:

```
procedure TForm1.BitBtn1Click(Sender: TObject);
var
i:Integer;
begin
TableTransportDataSet.Empty;
for i:=0 to memo1.lines.count-1 do begin
 Query1.Close;
 Query1.Sql.text:='Select * from '+memo1.lines[i];
 Query1.Open;
 TableTransportDataSet.Append;
 TableTransportDataSet.FieldByName('TableName').AsString:=memo1.lines[i];
 TableTransportDataSet.FieldbyName('Data').AsString:=CloneDataSetToString(Query1);
 TableTransportDataSet.post;
end;
ExtractButton.Visible:=TableTransportDataSet.recordcount>0;
end;
```

And this code shows how to use it.

```
procedure TForm1.ExtractButtonClick(Sender: TObject);
begin
DataSource2.DataSet.Free;
DataSource2.DataSet:=StringToDataSet(TableTransportDataSet.FieldByName('Data').AsString);
end;
```

1.4.3.4 Large FileTransfer

AstaParamlist messaging is easy to use and efficient but sometimes you may want to transfer very large files and don't want your Database server to be overwhelmed with file transfer requests. Of course you can always use FTP which WAS designed for file transfer and using your ASTA server to manage the files you want with ASTA messaging is a very efficient way to transfer files.

ASTA does offer an alternative with routines that can transfer files from server to client or client to server in configurable "chunks" or block sizes. With ASTA messaging typically files are loaded in TMemoryStreams, in the ASTA segmented file support calls, a TFileStream is used on both client and and server to use as little memory as possible.

Server to Client
Client to Server
Param List Large File Transfer

1.4.3.4.1 Standard Messaging

This example shows how to send the Client.exe itself to and from the server using several ASTA messaging techniques and also showing a progress bar when a file is requested from the server.

Server Side

```
procedure TForm1.AstaServerSocket1OutCodedParamList(Sender: TObject;
  ClientSocket: TCustomWinSocket; MsgID: Integer; InParams,
  OutParams: TAstaParamList);
var
  FileName:String;
begin
case msgid of
2000: begin
          FileName:=InParams[0].AsString;
          memo1.lines.add(FileName);
          OutParams.FastAdd(Filename,'');
          OutParams[0].loadFromFile(FileName);
```

```
end;
end;
end;
```

Client Side

```
Procedure TForm2.BitBtn2Click(Sender: TObject);
var
  pout, PIn: TAstaParamList;
begin
  Pout := TAstaParamList.Create;
  pout.FastAdd(Edit1.text);
  try
    pin := AstaclientSocket1.SendGetCodedParamList(2000, Pout);
    memo1.lines.clear;
    if pin.count > 0 then begin //found it
     memo1.lines.add('File Received of '+IntToStr(Length(pin[0].AsString))+' bytes');
    end;
  finally
    pin.Free;
    Pout.free
  end;
end;
```

1.4.3.4.2 Server to Client

Declaration

Description

1.4.3.4.3 Client to Server

With this demo, any file or file masks of files can be transferred from the client to the server in configurable chunks. Below is a screen shot of the server log showing all the AstaIO*.pas files being transfered with one file of code.

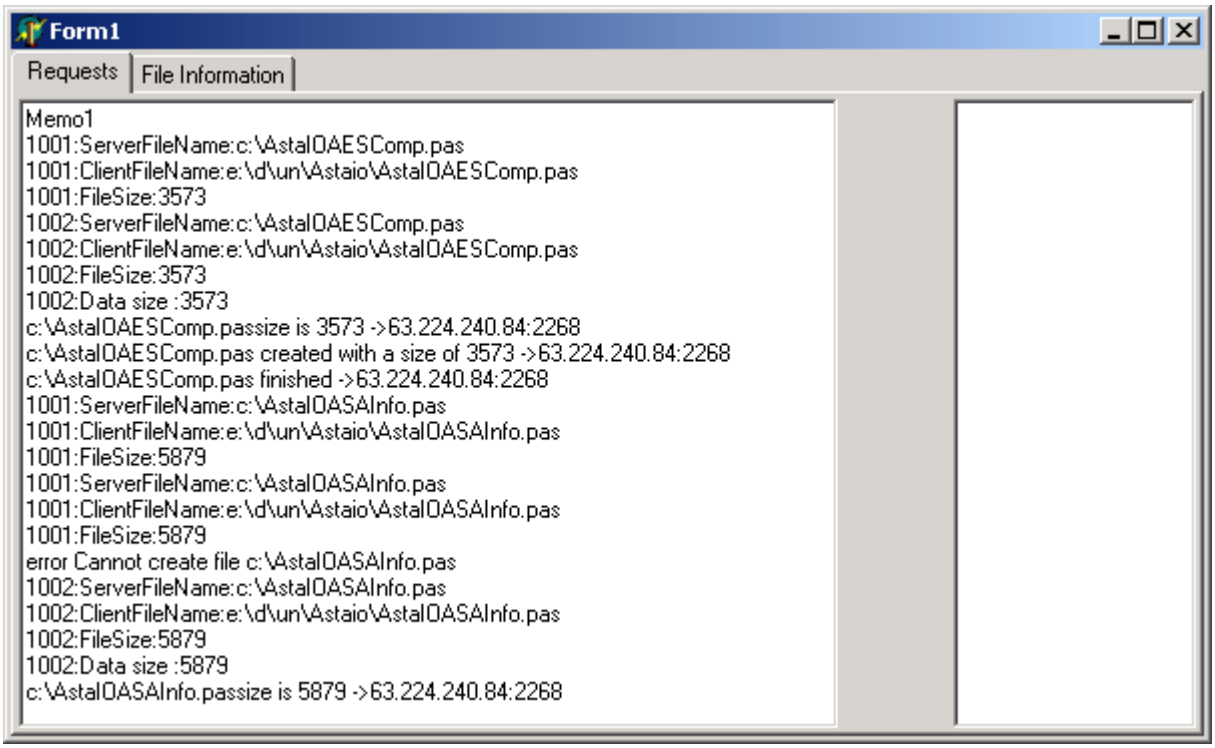
AstaServerFileSegment.pas implements a TAstaServerFileReceivelist component.

FServerFileList:=TAstaServerFileReceiveList.Create(ServerSocket);

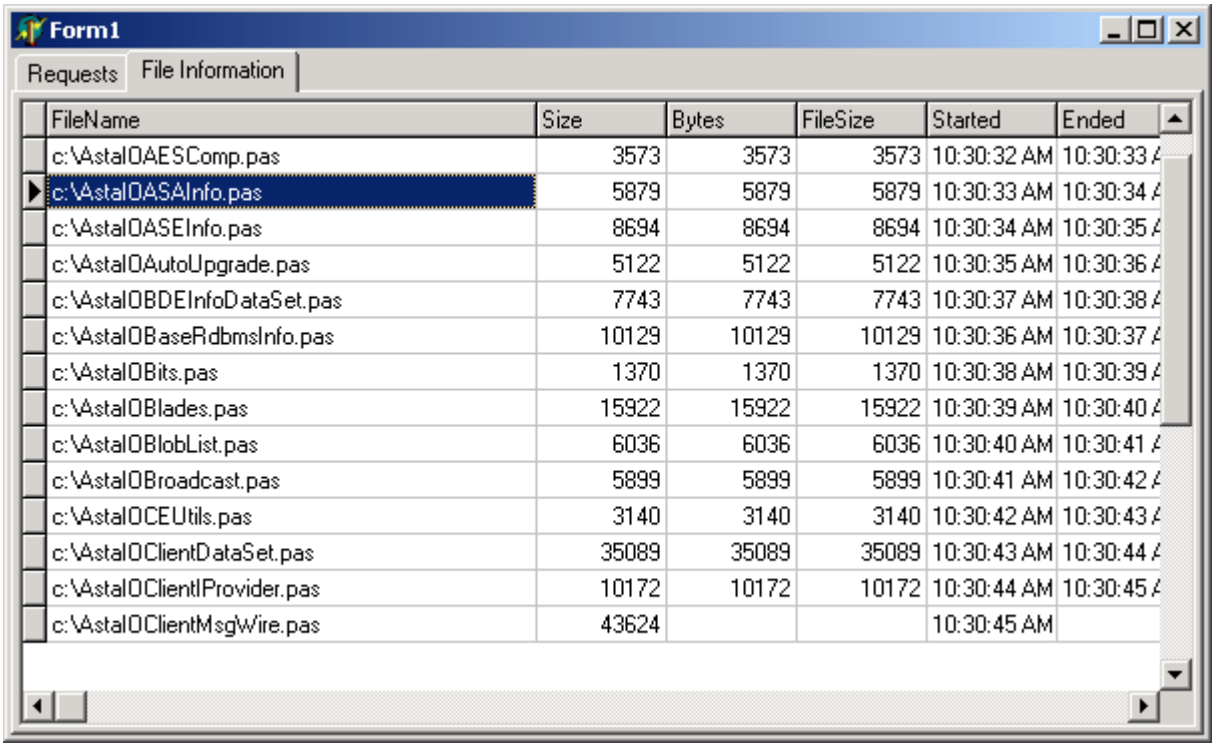
The code on the server is mostly to visually show the files coming in and being saved. To add support on the server to support client to server file transfer you just need to add a couple of lines to the AstaServerSocket.onCodedParamList event.

```
procedure TForm1.ServerSocketCodedParamList(Sender: TObject;
  ClientSocket: TCustomWinSocket; MsgID: Integer; Params: TAstaParamList);
begin
   Case Msgid of
     1001:FServerFileList.SetupFile(ClientSocket,Params);
     1002:FServerFileList.AddToFile(ClientSocket,Params);
    end;
end;
```

The Client Side code shows how to send one file or a file mask of files to the server very efficiently.



Below is a shot from the server log showing the actually sizes and more info as files come in.



1.4.4 Server side Techniques

In the N Tier Architecture it is common practise to put the "Business Logic" on the middle tier or ASTA server. These demos show a couple of techniques for using TAstaProviders and ServerMethods via the TAstaBusinessObjectsManager.

Provider and ServerMethod Text File Updater	Shows a non-database example, with a server and a cli of how to "query" directories on the server via a ServerMethod using the TAstaBusinessObjectsManager to "update" text files using AstaClientDatasets and TAstaProviders
Threaded Server Method Server	This examples shows a threaded server that handles an memory DataSet but has no real Database. The client w send
Server Method Parameter Example	Includes a non-database server with servermethods and shows how to return params from a servermethod. The example actually can stream down the server EXE itself a Param to the AstaclientDataSet

1.4.4.1 Provider Text file Updater

This samples includes both a server and a client. The server is not a database server but uses an Servermethod, using the TAstaBusinessObjectsManager, to return a list of text files from a Parameter defined on the serverMethod. It also uses a Provider to allow for the text file to be updated.

This example shows a simple of example of putting the "Business Logic" on the middle tier. There is no real database but we can use ASTA to feed data to remote clients by populating in memory DataSets on the server and intercepting applyupdates calls to write files back to disk rather than applying SQL to a Database.

Server Side
Since this server will use ASTA Server side components and not use messaging or ClientSide SQL, the Datamodules and forms used on the server must be registered. The call to TAastaServerSocket.RegisterDataModule "takes inventory" of existing Providers and ServerMethods

```
procedure TForm1.FormCreate(Sender: TObject);
begin
  Application.OnException:=AstaException;
  ServerSocket.RegisterDataModule(Self);
  ServerSocket.RegisterDataModule(ServerDM);
  ServerSocket.Active:=True;
end;
```

The ServerDataModule defines one ServerMethod that has a String Param of "FileMask". From the client, an AstaClientDataSet will point to the Servermethod and send a file mask to the server which will then execute the following code to use the Delphi Directory Find Routines to find all files matching the FileMask and optionally Load the file itself. The FileDataSet is an AstaDataset, an In memory DataSet with the fields already defined. The Servermethod will append the Dataset.

```
procedure TServerDM.AstaBusinessObjectManager1Actions0Action(
  Sender: TObject; ADataSet: TDataSet; ClientParams: TAstaParamList);
var
 FSearchRec:TSearchRec;
```

```
begin
  FileDataSet.open;
  FileDataSet.Empty;
  if sysutils.FindFirst(ClientParams.ParamByName('FileMask').AsString, 0, FSearchRec) = 0 then
     repeat
      with FileDataSet do begin
        Append;
        FieldByName('FileName').AsString := ExtractFileName(FSearchRec.Name);
        FieldByName('Size').AsInteger := FSearchRec.Size;
        FieldByName('TimeStamp').AsDateTime := FileDatetoDateTime(FSearchRec.Time);
        FieldByName('Path').AsString := ExtractFilePath(expandFileName(FSearchRec.Name));
        if ClientParams.ParamByName('LoadFiles').AsBoolean then
         TBlobField(FieldbyName('File')).LoadFromfile(ExtractFilePath(ClientParams.ParamByName
('FileMask').AsString)+FSearchRec.Name);
        Post;
      end;
      until sysutils.findnext(FSearchRec) <> 0;
      if FileDataSet.RecordCount>0 then FileDataSet.First;
end;
```

Client Side

The Client has an AstaClientDataSet but it does not use sql but points to the TAstaProvider on the server. If the user wants to bring the full text file over along with the Filename,Size, Timestamp etc they just click on a check box and set the 'LoadFiles' param.

```
procedure TForm2.Button1Click(Sender: TObject);
begin
with AstaClientDataSet1 do begin
  ParamByName('LoadFiles').AsBoolean:=LoadFilesCheckBox.Checked;
  ParamByName('FileMask').AsString:=FileMaskEdit.Text;
  //ASTA wont' bring over blob or memo fields unless you explictly ask for them
  // in this case it's a blob field so we set SQLOptions.soFetchBlobs to true
  if LoadFilesCheckBox.Checked then
   SQLOptions:=[soFetchBlobs]
  else SQLOptions:=[];
  Close;
  Open;
end;
end;
```

If we has the LoadFilesCheckBox.checked, the actually Text file is streamed over as part fo the DataSet "File" field. Now that the AstaClientDataSet is populated the text file shows in a DBMemo and we can edit the actual text file. To post it to the server, we just call

```
with AstaClientDataSet1 do
 ApplyUpdates(usmServerTransaction);
```

And 2 Datasets sent to the server that represent the current rows and "original" rows. On the TAstaProvider on the server we cannot use SQL since we are just editing a text file and there is no database. Here is the code on the server that will handle the updates. On the TAstaProvider there is an OnBeforeUpdate event that has a **VAR** Handled: Boolean that can be set if you don't want ASTA to generate SQL, which of course is not going to work so well in this case with no database on the server.

```
procedure TServerDM.TextFileProviderBeforeUpdate(Sender: TObject;
  ClientSocket: TCustomWinSocket; ExecQuery: TComponent;
  OriginalValueDataSet, CurrentValueDataSet, ServerValueDataSet: TDataSet;
  var Handled: Boolean);
        Function FullFileName:String;
        begin
         with CurrentValueDataSet do
```

```
result:=FieldByName('Path').AsString+Fieldbyname('FileName').AsString;
          AstaBusinessObjectManager1.actions[0].serverSocket.recordServerActivity(ClientSocket
,result);
         end;
begin
handled:=True;
with CurrentValueDataSet do
 if fileExists(FullFileName) then
  TBlobfield(FieldByname('File')).SaveToFile(FullfileName);
end;
```

1.4.4.2 ServerMethod Param Example

This samples includes both a server and a client. The server is not a database server but uses an Servermethod, using the TAstaBusinessObjectsManager. There are 2 Methods Defined: one called TestMethod and the other called GetFile, neither of which return a result set but only return params. ServerMethods that return no result sets will be able, in future ASTA 3 builds, to be published as SOAP Services by just setting the TAstaActionItem.SoapService:Boolean to true.

Method 1 : TestMethod

TestMethod defines 3 params.

IntputParam : ftSmallInt, ptinput
OutputParam : ftSmallInt,ptoutput
TimeStamp : ftDateTime,ptputput

In the OnActionEvent of the TAstaActionItem here is the code that receives the InputParam from the client and just sets the output Param. The ClientParams : TAstaParamList comes in from a remote TAstaclientDatset.

```
procedure TServerDM.AstaBusinessObjectManager1Actions0Action(
  Sender: TObject; ADataSet: TDataSet; ClientParams: TAstaParamList);
begin
  ClientParams.ParamByName('OutputParam').AsInteger :=
ClientParams.ParamByName('InputParam').AsInteger - 100;
  ClientParams.ParamByName('TimeStamp').AsDateTime := now;
end;
```

Here is the Client Side Code using an AstaClientDataSet with a ServerMethod pointed to TestMethod.

```
procedure TForm1.ReOpenClick(Sender: TObject);
begin
with AstaclientDataset1 do begin
 ParambyName('InputParam').AsInteger:=StrtoInt(Edit1.text);
 refiresql;//this does a disable control/close/open/enable controls
end;
end;
```

Method 2 : GetFile

GetFile defines 2 params.

FileName : ftstring, ptinput
File : ftblob, ptoutput

In the OnActionEvent of the TAstaActionItem here is the code that receives the InputParam from the client which in this case is just a filename. If that file is found on the server it is streamed back in the output Param otherwise an exception is raised which will propogate back to the client.

```
procedure TServerDM.AstaBusinessObjectManager1Actions1Action(
  Sender: TObject; ADataSet: TDataSet; ClientParams: TAstaParamList);
begin
  if fileExists(ClientParams.ParamByName('FileName').AsString) then
    ClientParams.ParambyName('File').LoadFromFile(ClientParams.ParamByName('FileName').AsStrin
g)
  else raise Exception.create(ClientParams.ParamByName('FileName').AsString + ' does not
exist');
end;
```

Calling ServerMethods at runtime

The demo also has an example of calling servermethods at runtime. When the serverMethod name is set, the params are fetched from the server so you don't need to create the params. This happens at design time also when an AstaClientDataSet is set to use a Provider or ServerMethod. For performance purposes you can cache all the Param info for Providers and ServerMethods so there is no need for a trip to the server.

```
procedure TForm1.Button2Click(Sender: TObject);
var
  acd: TAstaClientDataSet;
begin
  acd := TAstaClientDataSet.Create(nil);
  try
    acd.AstaClientSocket := AstaClientSocket1;
    Acd.ServerMethod := 'ServerDM.GetFile';
    acd.ParamByName('FileName').AsString := FileEdit.Text;
    acd.Open;
    showmessage('File Size is ' + IntToStr(Length(acd.ParambyName('File').AsString)));
  finally
    acd.free;
  end;
end;
```

1.4.4.3 Threaded Server with No Database

This samples includes both a server and a client. The server is not a database server but uses an Servermethod, using the TAstaBusinessObjectsManager. This Server is threaded so you can see the minimum required to thread Servermethods with ASTA servers. There is no real database but instead there is a servermethod that defines a Parameter named "RecordsToCreate" so the client determines how many rows to create on the server. An in memory DataSet, the TAstaDataSet is used on the serverMethod which supplies the result set streamed down to the AstaClientdataSet.

When the server starts up, any form used on the server needs to be registered so that ASTA can inventory the server side components. This is done by calling the AstaServerSocket.RegisterDataModule method. Then CreateSessionsforDBThreads is called, ThreadedDBSupplySession is called Database Sessions number of times to create a "pool" of DataModules. Typically the main DataModule is a DataBase DataModule with some kind of Database Connection Component like a TADOConnection or TDatabase/TSession pair. When a user connects to an ASTA server no threads are

required. When any request for a datbase process is made, a "session" is checked out of the thread Pool, the work is done in a thread, and the Session is returned to the Pool as early as possible. To scale servers, get resources late and release them early. ASTA does all this for you if you just set a couple of properties and code some events.

Threading the Server
ASTA Database servers already come thread ready but since this is not a database server there are a couple
of steps you need to do to thread the server.

1. Set the AstaServerSocket.ThreadingModel to tmSmartThreading
2. Set the AstaServerSocket.DatabaseSessions:integer to a value greater than zero.
3. Code the AstaServerSocket.ThreadedDBSupplySession Event.
4. Code the AstaServerSocket.ThreadedDBSupplyQuery event.
5. Call the AstaServerSocket.CreateSessionsForDBThreads method to set up the Database Session Pool.

Here is the code from the server that sets the server up for threading.

```
procedure TForm1.FormCreate(Sender: TObject);
begin
  Application.OnException:=AstaException;
  ServerSocket.RegisterDataModule(Self);
  ServerSocket.RegisterDataModule(ServerDM);
  ServerSocket.Active:=True;
  ServerSocket.CreateSessionsForDBThreads(True);
end;
```

Server Method : DataSetRows
A ServerMethod is defined using a TAstaBusinessObjectsmanager. On this server there are 2 TAstaBusinessObjectsManagers to show you that you can use as many as you wish and you can also use as many DataModules as you need on the server also. The server defines one Param caled RecordsToCreate that is an Input Param and of type Integer.

The code below opens and Empties the TestDataset:TastaDataSet and then just loops through and adds as many rows as defined on the client incoming Param.

Note: When populating your own DataSet, ASTA will not call FIRST on the server for performance considerations. so if you are populating your own in memory DataSet remember to call First.

```
procedure TServerDM.AstaBusinessObjectManager2Actions0Action(
  Sender: TObject; ADataSet: TDataSet; ClientParams: TAstaParamList);
var
   LoopCounter,Total: Integer;

begin
     TestdatASet.Open;
     if testDataSet.RecordCount>0 then
     TestDataSet.Empty;
     total:=ClientParams.ParamByName('RecordsToCreate').AsInteger;
     with TestDataSet do
     for LoopCounter := 1 to Total do
     begin
      Append;
      FieldByName('StringField').AsString := 'This is a test field index : ' +
IntToStr(LoopCounter);
      FieldByName('IntegerField').AsInteger := LoopCounter;
      Post;
     end;
```

```
TestDataSet.First;
end;
```

Client Side

On the client side, there is an AstaClientDataSet with the serverMethod set to the DataSetRows ServerMethod on the server. The Only thing required is that you set the AstaClientDataSet.Params and then close and open the DataSet. The params will go to the server, check out a datamodule, execute in a thread, and call the server method OnActionEvent.

```
procedure TForm1.Button1Click(Sender: TObject);
begin
with RowDataSet do begin
 Params[0].AsInteger:=StrToIntDef(Edit3.Text,25);
 RefireSQL;
end;
end;
```

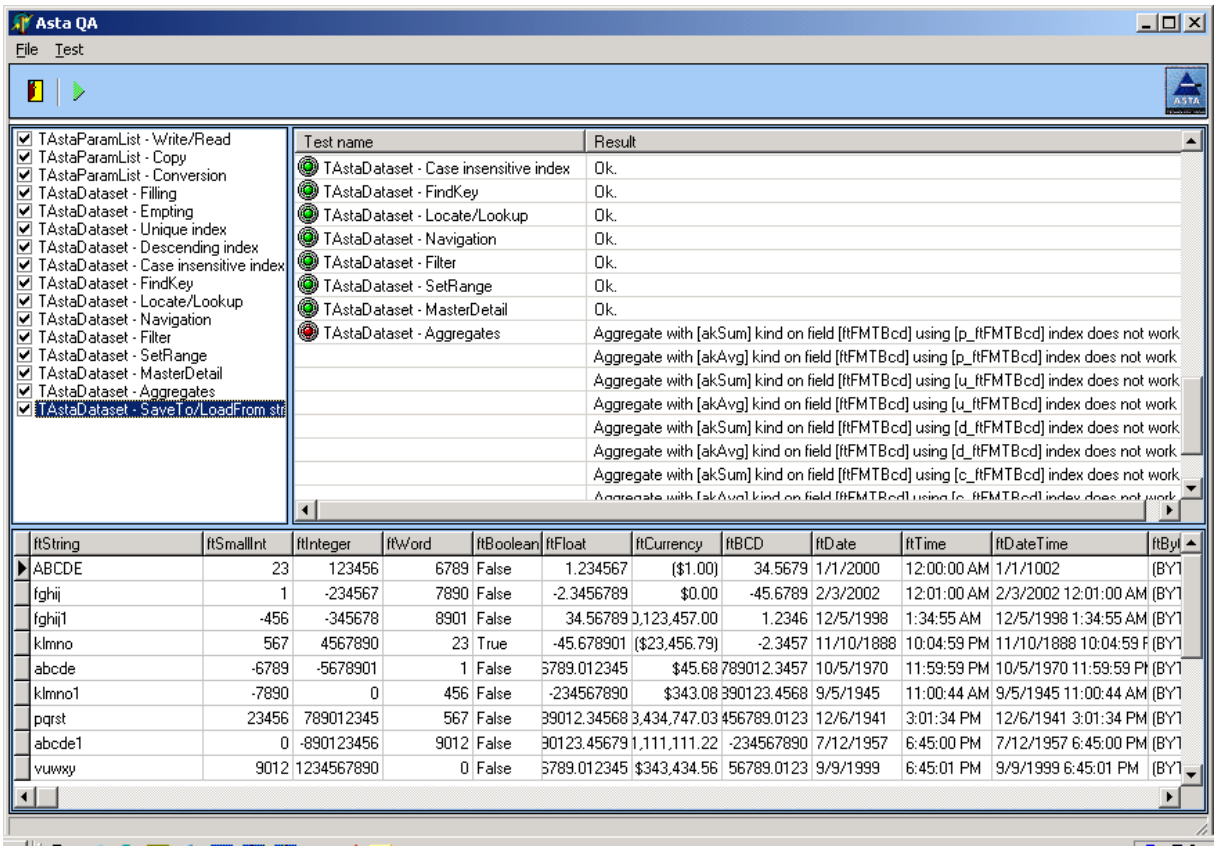
1.4.5 Feature Testers

These tutorials also can be used to test and adminster remote servers. It's not a bad idea to compile them and install them in the Delphi IDE until Tools Option in order to fire SQL against any remote server, see all your ServerMethods or providers on remote servers or see everything you want with the AstaSQLExplorer. Remember these are all ASTA thin client apps that can be used against any remote server.

DataSet QA
Provider Tester
ServerMethod Tester
SQL Demo and http Tunneling
SQL Explorer
Stored Procedure Tester

1.4.5.1 DataSetQA

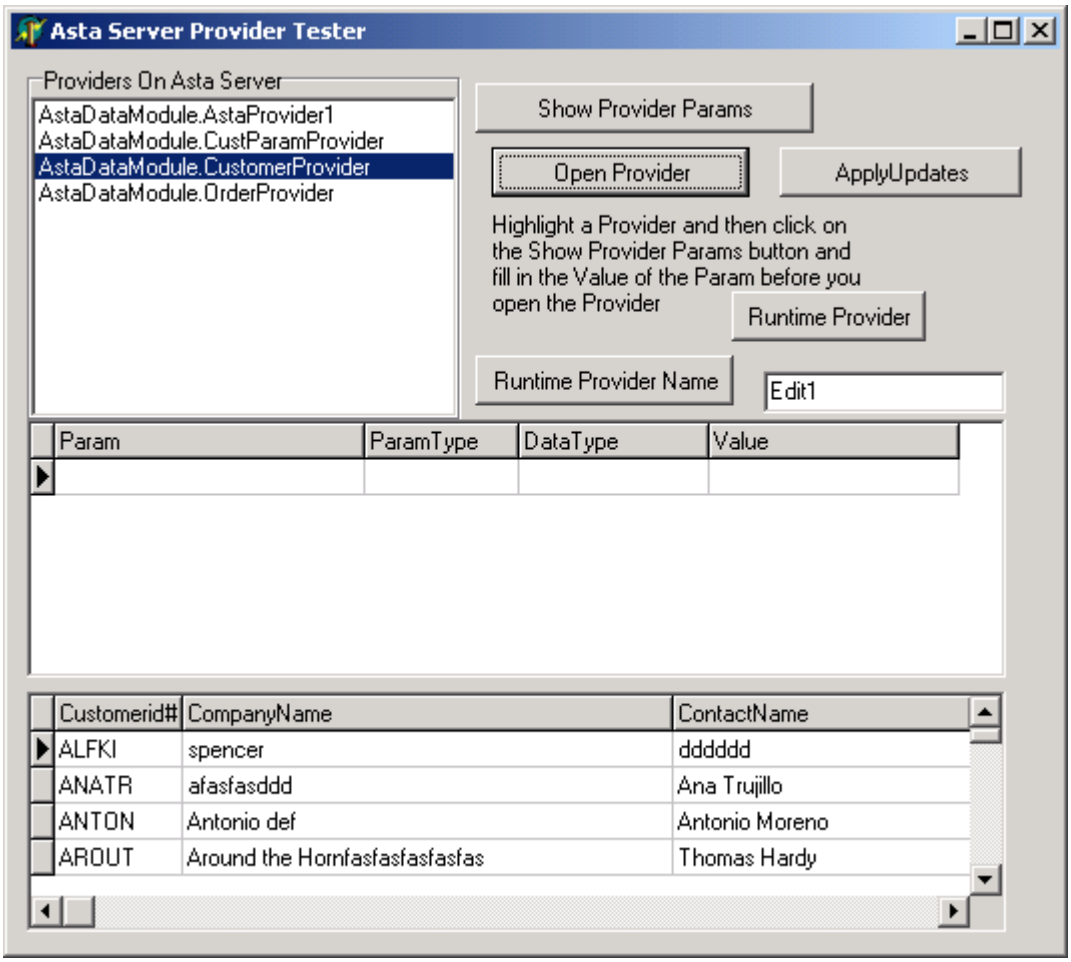
We've included one of our QA tests in the ASTA 3 tutorials to show how ASTA properly handles all datatypes. In fact we've even found a bug we have reported to Borland.



1.4.5.2 ProviderTester

Business Logic can be implemented on the middle tier with ASTA by using the TAstaBusinessObjectsManager component or TAstaProviders. The AstaProviderTesterTester shows all the providers running on a remote ASTA server, the params for those providers and allows the params to be set and a result set requested.

The result set can be edited and ApplyUpdates called to post the changes to the server.



1.4.5.3 ServerMethodTester

Business Logic can be implemented on the middle tier with ASTA by using the TAstaBusinessObjectsManager component. The AstaBusinessObjectsTester shows a couple of techniques that can be used with server methods

This client can be run against any ASTA server and will show all the available ServerMethods and the params for any given servermethod. Params can be set and ServerMethods Executed.

When an AstaClientDataSet has it's servermethod property changed, either at runtime or design time it will bring back param information for the method choosen. This param information can come from the server or can be cached on the client to avoid extra server round trips. The AstaclientSocket has a CachedMetata property that allows you to cache param and other info (like provider primekey info) for TAstaProviders and ServerMethods. It is recommended that this be set to smFetchOnValidate.

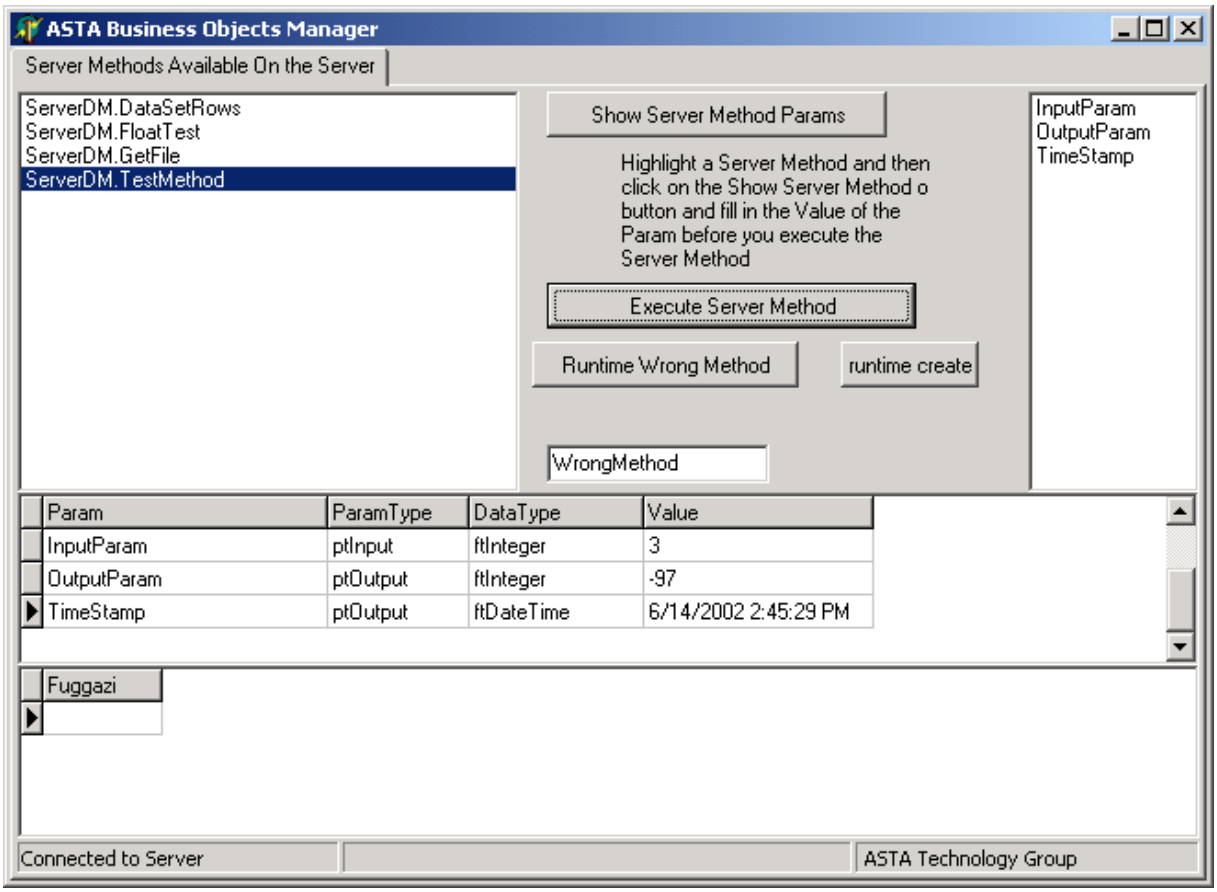
Showing Available Server Methods
The TAstaClientDataSet has a MetaDataRequest propety that allows for many kinds of information to be streamed over from remote ASTA servers in a DataSet. In this case this

property is set to mdBusinessObjects and will bring back a list of servemethods.

Show the Params for any ServerMethod
A Technique is show to get ParamInfo for each Param.

Executing any Servermethod
ServerMethods can be assigned and executed at runtime and this example show how to do that. There is also an example of showing how an exception is raised if a servermethod name assigned does not exist on the remote server.

Below is a shot of the ServerMethod Tester running against one of the ASTA tutorial servers.

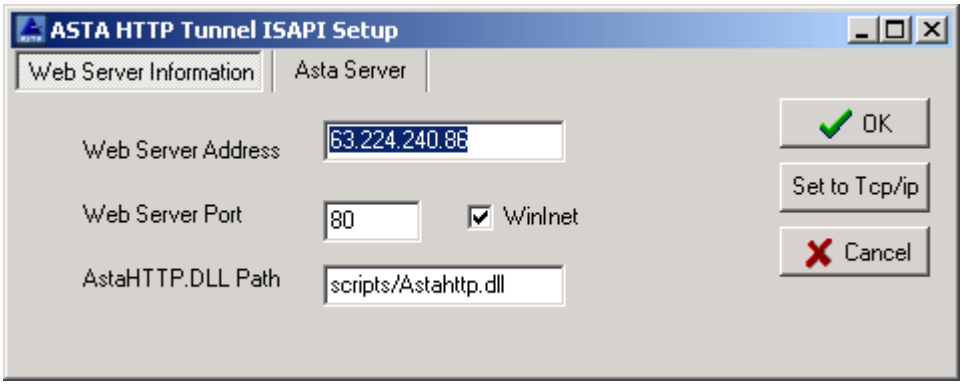


1.4.5.4 SQLDemo and Http Tunneling

The AstaSQLDemo allows you to connect to any ASTA server and get a list of table names, field info for each table and to SQL Select and Exec Queries. It also shows how to execSQL in a transaction. It is invaluable to test SQL against your database so it is

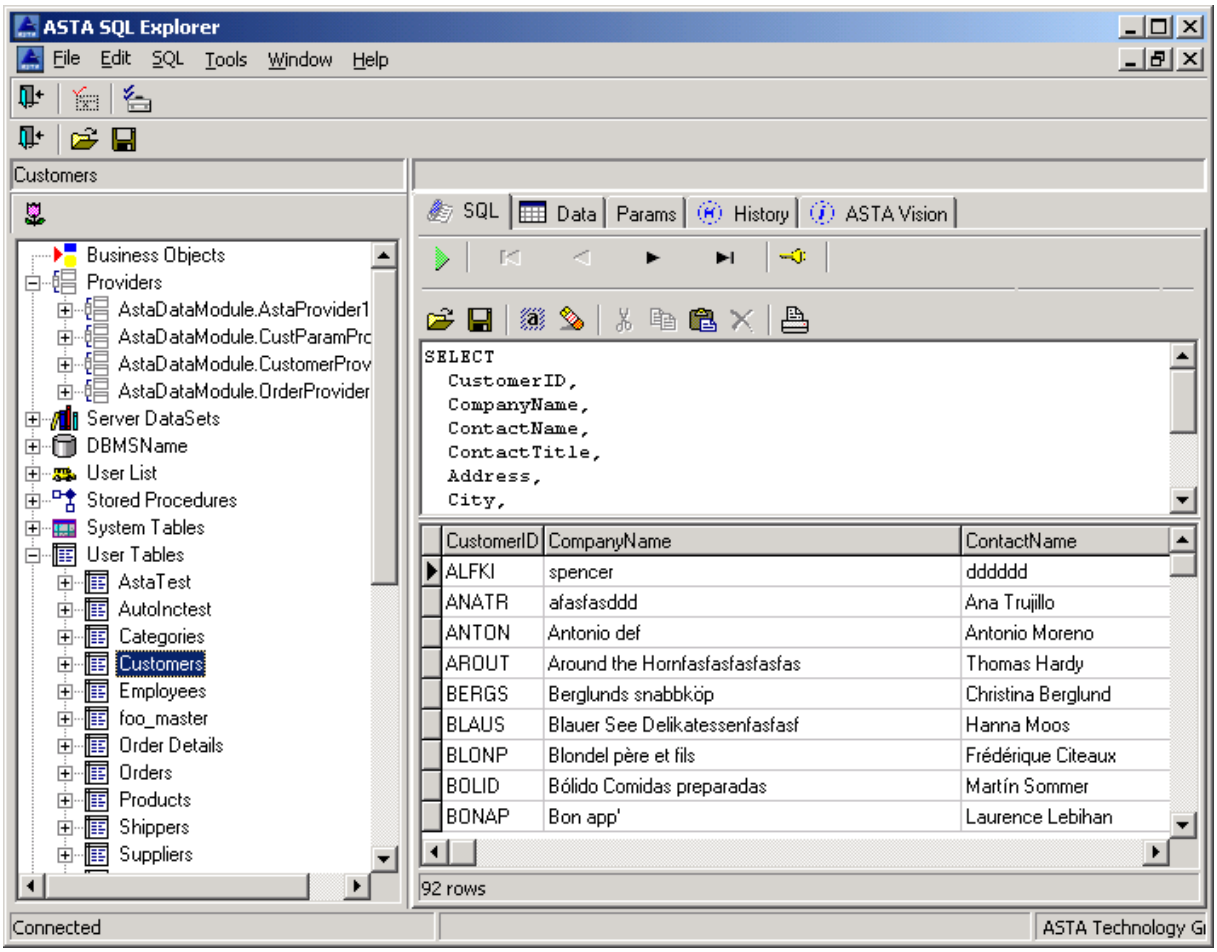
recommended you keep a compiled version around and even add it to the Options Menu choice in Delphi so you can run it at any time.

Additionally, it can be set to work with tcp/ip or http with a menu choice that allows you to set a web server, the location of Astahttp.dll and the location of the ASTA server.



1.4.5.5 SQLExplorer

The AstaSQLExplorer allows you to browse server side components and metadata information from your database. It will handle multple Datasources if your server supports them. The source is a good source for ASTA techniques in fetch server side inforamation. Of course it will run against any of the over 30 ASTA servers available using Delphi 3rd party Database Components.



1.4.5.6 Stored Procedures Tester

Declaration

Description

1.4.6 Security Issues

Declaration

Description

1.4.6.1 Servers and Security

Declaration

Description

1.5 ASTA and .NET

Asta .NET DataTable without .NET

With the release of .NET the Windows landscape has been forever changed. Microsoft has foresaken COM and, with the help of Turbo Pascal Creator and Delphi 1 Chief Architect Anders Heilsjsberg, Microsoft now has Delphi like Tools. At the Borland conference in Anahheim, Anders showed the new code to create a form in .NET. The old Windows Petzold style of coding required calls like this

```
HWND CreateWindowEx(

    DWORD dwExStyle,   // extended window style
    LPCTSTR lpClassName,      // pointer to registered class name
    LPCTSTR lpWindowName,     // pointer to window name
    DWORD dwStyle,     // window style
    int x,     // horizontal position of window
    int y,     // vertical position of window
    int nWidth,        // window width
    int nHeight,       // window height
    HWND hWndParent,   // handle to parent or owner window
    HMENU hMenu,       // handle to menu, or child-window identifier
    HINSTANCE hInstance,      // handle to application instance
    LPVOID lpParam     // pointer to window-creation data
   );
```

Anders showed new calls that looked like Delphi code and commented that this was not new for the Delphi guys and good ideas don't change!

```
Form1:=TFormCreate(Nil);
```

ASTA announces the ASTA .NET Rich Client Initiative.

http://msdn.microsoft.com/library/default.asp?url=/library/en-us/cpref/html/frlrfSystemDataDataTableClassTopic.asp

This summer, ASTA will release the first version of an AstaclientDataSet built with C++ that will run on Win32, WinCE, Palm and GCC Linux. Our first target platforms are Wince and Win32 and modeled on DOT Net.

These tools will be part of the ASTA SkySync initiative which provides for Cross Platform Syncronization.

http://www.astawireless.com/products/skysync/

Combined with the existing ASTA cross platform messaging, ASTA users will be able to connect to any remote database server built to run on Win32 or Linux or any platform that supports Java, disconnect from the server ,and then edit insert and delete as required. At a later time, the client will be able to connect once again to the remote server and post updates with full conflict resolution via Delphi or scripting.

By using the .NET model ASTA lays the groundwork for .NET clients without the necessity for the CLR. ASTA plans to support .NET clients with applications written using VB6, Delphi 5 and 6, Kylix, Java, Palm, WinCE and GCC Linux.

Java is best used on the server as is .NET. With thin, rich, no admin ASTA clients the best of both worlds can be achieved. Use Architect level programmers to code servers using Delphi, C# or Java and application devleopers using Delphi, Visual Basic 6 and any of the Popular PDA DEvelopment tools to create secure, thin client applications that run efficiently over the Internet.

This initiative will include virtualizing the ASTA Delphi and Kylix DataSets to be able to be configuarable as to the amount of memory used along with a full SQL92 client SQL parser. Persistency will be achieved using individual files or a paging system to allow for any number and amount of data to be streamed to one file. ASTA will implement this under Pascal and C++ for SkyWire Use.

The result will be full crossplatform supported persistent Data Stores with full SQL and XML support and the ability to syncronize to remote servers.

```
class CAstaDataTable : public
CAstaDataObject {
public:
  bool GetCaseSensitive(void) const;
  void SetCaseSensitive(bool AValue);
  DWORD GetMinimumCapacity(void) const;
  void SetMinimumCapacity(bool AValue);
  LPCSTR GetTableName(void) const;
  void SetTableName(LPCSTR AValue);

  CAstaDataColumnList * Columns(void) const;
  CAstaDataRowList * Rows(void) const;

  void Clear(void);
  void Reset(void);

  void ImportRow(CAstaDataRow * ARow);
  CAstaDataRow * NewRow(void);

  CAstaDataTable * Clone(void); // struct
  CAstaDataTable * Copy(void);  // struct & data

  void BeginLoadData(void);
  void EndLoadData(void);

  void AcceptChanges(void);
  void RejectChanges(void);
};
```

1.6 Building Applications with ASTA

1.6.1 General Topics

1.6.1.1 ASTA Jump Start Tips

1. Always code the AstaclientSocket.OnDisconnect event even with a comment.

2. Run an ASTA server unchanged first and write some Client side SQL before you start to code the server so you can get a "feel" for the way things work

3. Don't worry about writing Update, Insert or Delete SQL Statements as ASTA was designed so that you only need write Select SQL and then just set the EditMode property of the AstaClientDataSet.

4. Do not tried to open AstaClientDataSets in the formcreate method. The earliest you can open an AstaClientDataSet is in the OnConnectmethod of the AstaclientSocket.

5. If you want to reconnect an ASTA ClientSocket after it has disconnected, you cannot set the Active to true Property in the OnError or OnDisconnect Event. Use a TTimer instead to allow the event to be passed through and to have the timer trigger the Active := True outisde of the OnError or OnDisconnect event.

6. If you want to force a design time repopulation of an AstaClientDataSet, click on SQL Workbench and Design Populate. This is useful also if you have altered a table and want to force ASTA to go to the server at design time.

7. If you have an AstaClientDataSet open at design time do not try to open it in the OnConnect method. You can do one OR the other but not both.

8. NEVER set the AstaClientSocket.Active at design time. The Borland sockets that ASTA inherits from do not support design time connection.

9. To use the AutoClientUpdate feature you must have your clients log into the server.

1.6.1.2 ASTA Compression

To use compression with ASTA just set the Compression property of the AstaServerSocket and AstaClientSocket to AstaZLib. All data now passed between clients and servers will be compressed. AstaZLib compression is available cross platform including Win32, Palm, WinCE and when used with ASTA Java clients.

AstaZLib compression is threadsafe and full Pascal source is available.Thanks to ASTA user David Martin for his modifications to this unit. AstaZLib compression is implemented in AstaPasZLibCompress.pas.

```
procedure ZLibCompressString(var s: string; compressionLevel: Integer);
{ Compress string s using zlib (thread-safe). This is based on the same
  algorithm used in PKZip. compressionLevel may vary from 1 (best speed
  but worst compression) to 9 (slowest speed but best compression). A
  compressionLevel of 1 will usually give good speed with good
  compression. However, a higher compression level may yield better
  performance when bandwith is at a premium because the total data
  transfer time is given by the sum of compression time + download time
  + decompression time. }
```

```
procedure ZLibDecompressString(var s: string);
{ Decompress string s that was compressed using ZLibCompressString. }
```

Note: earlier versions of ASTA used a freeware lzh compression unit that turned out not to be thread safe. AstaZLib compression is recommended.

1.6.1.3 Registry Keys Used by ASTA

Main Key:
AstaRegKey = 'HKEY_CURRENT_USER\SOFTWARE\Asta'.

IP address as stored from the FastConnectCombo:
HKEY_CURRENT_USER\AstaRegKey\Misc\IPforTesting

Disable/Enable AutoDial (use 0 or 1):
HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Internet Settings\EnableAutodial

```
procedure DisEnableAutoDial(bDisable: Boolean);
var
  Reg: TRegistry;
begin
  Reg := TRegistry.Create;
  try
    Reg.RootKey := HKEY_CURRENT_USER;
    if Reg.OpenKey('\Software\Microsoft\Windows\CurrentVersion\Internet
  Settings', False) then
      //False = don't create if not found!
      begin
        if bDisable then
          Reg.RenameValue('EnableAutodial', 'EnableAutodial_XXX_Temp')
        else
          Reg.RenameValue('EnableAutodial_XXX_Temp', 'EnableAutodial');
        Reg.CloseKey;
      end;
  finally
    Reg.Free;
  end;
end;
```

1.6.1.4 Useful Utility Methods

Unit
AstaUtil

Here are some helper functions used internally by ASTA:

```
procedure StringToStream(const S: string; var TM: TMemoryStream);
function NewStringToStream(S: AnsiString): TMemoryStream;
function StreamToString(MS: TMemoryStream): AnsiString;
function GetThePCsIPAddress: string;
```

1.6.2 Database Discussion

1.6.2.1 Client Side SQL vs Coding the Server

The fastest way to deploy internet ready client server applications is to use ASTA's client side SQL. This allows you to use ASTA servers unchanged and to write ASTA client applications much the same way you would code normal 2 tier fat client applicatons.

If your existing BDE applications have TQueries on data modules then you simply need do a Save As and create a new data module. Then replace all the TQueries with TAstaClientDataSets and the TDatabase with a TAstaClientSocket. If your application already has good client/server manners then you should be able to ASTA'cize your application very quickly. There is an ASTA Conversion Wizard to help out with this process.

N-tier programming purists believe that there should be NO SQL on the client and all the "business rules" should be defined on the server. ASTA supports this also with TAstaProviders, TAstaBusinessObjectsManagers and ASTA messaging. This allows you to deploy ASTA clients that use NO SQL but merely point to either a TAstaProvider on an ASTA server or a server side method.

To use ASTA server side components you must of course code an ASTA server, so development time may be greater than with client side SQL. However there are other benefits to deploying clients with NO SQL in that the client application need not change with modifications only made to the server.

It is considered "Best Practice" to use an Application server to centralize the business logic and increase security. Even if you are developing a Web Application via an ISAPI DLL it is recommended that that you don't communicate directly to the database via SQL but instead create a de-militarized zone (DMZ) where you only call Remote Procedures from your web apps and allow an application server to implement the business logic. In addition, as the Application Server will pool database connections you add scalibility as well as security to your design.

Once you have business logic on an application server you may then "share" some of those Methods with business partners, by publishing them via SOAP Services so that your business partners can have access to required services.

1.6.2.2 Packet Fetches

AstaClientDatasets can be set to Open a Server side cursor and return RowsToReturn number of rows. In this way large queries can be more efficiently fetched over latent lines. In order to use this technique the ASTA server must be running in PersistentSessions threading model which requires each remote client to have their own Database Session. Although this technique is useful, the AstaServer will not scale as well as when it is run using the recommended Pooled Threading Model.

When RowsToReturn is set to -1 all rows are returned. When RowsToReturn is set to 0 only the FieldDefs are returned and an empty DataSet is returned. This may be useful when only inserted rows are required.

The AstaClientDataSet.SQLOption of soPackets must be set to true to enable Packet Fetches. When the initial select is executed on the server, RowsToReturn number or rows are returned. To retrieve subsequent rows, you must call GetNextPacket. There is an

AutoFetchPackets property on the AstaClientDataSet that if set to true, will attempt to call GetNextPacket if any Visual Controls are connected to the DataSet like a TDBGrid.

When the AstaClientDataSet is closed CloseQueryOnServer is called to close the TDataSet component used on the server for the select. You may also call CloseQueryOnServer directly if you want to close the server side TDataSet. The component used for this is determined by a server side call to ThreadedDBSupplyQuery which must be coded to dynamically create a TDataSet for a client packet fetch request.

After a Packet Fetches is executed you may also then do a GetNextPacketLocate which executes a server side Locate call on the open TDataSet on the server. Performance will depend on the size of the result set and how the Locate is implemented by your server side component. File server components like DBISAM can use indexes on locates so performance can be quite fast. For client/server databases indexes are not available on result sets so server side locates would have to iterate sequentially through the DataSet so large result sets may not result in fast results.

1.6.2.3 Client/Server Manners

The two most important factors affecting performance of your ASTA client application are bandwidth and your application design. It is important to follow the client/server prime directive: Fetch ONLY what you need WHEN you need it.

This means that: SELECT * FROM Customers

Becomes: SELECT Last, First, ID FROM Customers WHERE Age > 30

If you come from a file server background, then you should think about moving from a procedural, navigational model to an SQL result set oriented design. The "appearance" of great speed and responsiveness can be achieved by limiting the amount of data requested from ASTA servers, and the number of requests made to the server. Client/server round trips are costly.

See also:
The Paradox of Databases Online
ASTA for Database Developers Online

1.6.2.4 Distributed Database Networks

ASTA Database Replication Engine
ASTA allows remote users to connect to databases across the Internet and efficiently communicate with a remote server. When deploying applications over WAN's and the Internet, developers must be very conscious of the cost to and from remote servers. In this brave new world the most precious resource is now bandwidth, not cpu speed, memory or database speed.

Disconnected Users
Oftentimes users work remotely or connect to a server for a limited period of time and then want to be able to work "offline". With the rise in wireless communication and the spread of reasonably priced connectivity, more and more employees will be given the opportunity to work remotely but will not always be able to be connected to a remote

server 24 hours a day, 7 days a week.

ASTA Suitcase Model
ASTA supports a suitcase model where users can fetch data from an ASTA server and then disconnect from the server, modify the data, save it to a file, and then come back at a later date, reconnect to the server and apply any inserts, updates or deletes. This suitcase model can be complicated by the fact that data on the server may have changed during the time the user was disconnected.

Disconnected Groups of Users
Sometimes groups of users may work remotely and be connected by a local area network or run their own ASTA server. These users may need to share data and that data may need to be synchronized with a remote Database of other users also sharing Data.

Fairly Static Data
Sometimes data that a remote client selects from a server does not change too often. Some ASTA users have taken to performing some queries when an ASTA server starts up or at fixed intervals and then compressing that data and streaming it down to remote ASTA clients using ASTA messaging. This data needs to be efficiently "refreshed" and synchronized between host and remote sites.

The Solution
To deal with all the above issues, ASTA is developing an Add On that uses the internal code name of AstaPosBus. (PosBus is Afrikaans for Post Office Box and was coined by the PosBus project leader Stephan Marais as Stephan had experience on a simlar project in his native South Africa). ASTAPosBus will allow table by table and field by field replication and syncronization. Replication can be very complicated subject but ASTAPosBus will attempt to provide base methods to allow for ASTA developers to easily synchronize data between remote databases using ASTA servers.

Purpose
To provide a basic process to sync remote databases with a server database. This does not apply to database schema changes, only data changes. There are 3 different actions that can be applied to a table, and depending which kind of action, it must be handled differently.

Insert
Can be handled in two ways:
1. The table must contain an update_date/insert (datetime) field
2. An audit table can be used to record the insert. The audit table must also contain an "action" field

Update
Can be handled in two ways:
1. The table must contain an update_date/insert (datetime) field
2. An audit table can be used to record the insert. The audit table must also contain an "action" field

Delete
1. An audit table can be used to record the insert. The audit table must also contain an "action" field

Assumptions
1. Primary keys are not allowed to be changed. If needed, delete the record and make a

new entry
2. The tables on the server must have at least all the fields as the tables on the client
3. If audit tables are used, each audit table must have a field called action_indicator (char) and audit_date (datetime)

1.6.2.5 Suitcase Model

ASTA supports a suitcase or persistent model in that any TAstaClientDataSet can:

1. Save and load itself from a stream
2. Save and load itself from a file
3. Save and load itself form a blob field (since it can stream itself)
4. Sort itself on any column (except blob fields) or multiple columns
5. Use it's built in filter to simulate indexes

The following discussion uses the example as supplied with the ASTA components in the \ASTA\Tutorials\SuitCase directory.

The TAstaClientDataSet has a property called SuitCaseData that expands to sub properties:

Active	This determines whether the TAstaClientDataSet will fetch data from a local disk or connect to an ASTA server for it's data.
FileName	This is a name to be used by the TAstaClientDataSet to stream and load itself from disk.
SaveCachedUpdates	This also saves and loads the internal FOldValuesDataSet where ASTA stores the original values if EditMode is set to Cached or After Post.

In the Asta Suitcase tutorial, a normal query is run but then the TAstaClientDataSet saves the data from the query to a file by calling AstaClientDataSet1.SaveSuitCaseData. This saves all the data, including all the field types (even blobs and memo fields) to disk. Run the tutorial, do the query and click the Save to File button. Stop the tutorial, and this time set the SuitCaseData.Active property to True. When you compile and run the tutorial, instead of fetching the data from the remote ASTAServer, the TAstaClientDataSet will load the data from the SuiteCaseData.FileName property.

The following methods are available to aid in the Suitcase Model and are implemented in the TAstaDataSet the ancestor of the TAstaClientDataSet.

```
procedure SaveToStream(Stream: TStream);
procedure LoadFromStream(Stream: TStream);
procedure SaveToFile(const FileName: string);
procedure LoadFromFile(const FileName: string);
```

Power User Tip:
Sometimes when you want to use a DBGrid component on a form connected to a

TAstaDataSet you want the grid to show the field names as column headers even when the TAstaDataSet has not fetched any data. Follow these steps:

1. Set up a normal query with some SQL and set the Active property to True.
2. Set the Suitcase.Active to True but DO NOT put a file name in the Suitcase.Filename property. This will have the effect of keeping the table open at run time but NOT firing a query to fetch data from the server. This is what OpenNoFetch does.
3. When you want to fetch your data, instead of closing and opening the table, call the AstaClientDataSet.RefireSQL method.

1.6.2.6 Master Detail Support

The TAstaDataSet and the TAstaClientDataSet both suport master detail linking with MasterSource and MasterFields properties. When the AstaDataSet is used as a DetailDataSet it uses a filter to implement the limiting of visible rows. The AstaClientDataSet requires a parameterizd query to run live master detail queries from the server.When the Master changes a new query is fired to the server for the detail.

The following is an example of a detail query linking to a master customer table:

```
SELECT * FROM Orders WHERE CustNo = :CustNo
```

After writing the SQL be sure to set the Parameter datatype using the Params property of the AstaClientDataset. As the MasterDataSet is scrolled a new query is fired from the detail dataset using the linked field to populate the Param. On inserts you are required to code the OnNewRecord event of the DataSet to set any detail values that are linked to the master.If the Param name, in the above case Custno, matches the linked field on the server, then the Detail CustNo field will be automatically populated.

AstaClientDataSets can also run in masterdetail mode disconnected from the server by fetching the complete detail dataset and then using filters. There are AstaClientDataSet helper methods to facilitate moving back and forth between live detail selects using parameterized queries and the fetching of the complete detail table for all master fields using filters. See AstaClientDataSet. SetToDisconnectedMasterDetail and AstaClientDataSet. SetToConnectedMasterDetail and the BDE Master Detail tutorial for an example.

For support of AutoIncrement fields on Master DataSets that Detail DataSets link to there is a method called SendMasterDetailAutoIncTransaction.

1.6.2.7 Asyncronous Database Access

The Asta Pooled threading model has been extended to allow asyncronous query support. In ASTA's Pooled Threading Modela Pool of Database Sessions are created at system startup. Using the BDE as a model, this typically consists of a Datamodule that includes a Tdatabase/TSession Pair. On server start up you can say Sessions=8 and 8 Datamodules will be created when the server starts up.

When a remote ASTA client makes a database request, using a TAstaClientdataSet, the client goes into a wait state and sends the request to the server. On the server, one of

the "Sessions" is checked out of the Pool and a thread is launched with that session.This means that only one client side process can be executed at any time for a remote client.

In order to write code like:

```
AstaclientDataSet1.Open;
AstaClientDataSet1.Next;
etc.
```

ASTA must hide the asyncronous nature of sockets and go into a "wait state".

The client socket is used as a token on the server to "check" a session out of the pool. ASTA allows a remote client to have more than one Session in use on the server using ASTA messaging. So on the client you can now SendACodedParmList and on the server you can thread the message call using the ThreadedDBUtilityEvent to ask for a query, execute the sql and stream the result back in a TAstaParamList. The AstaBDEServer is now coded to allow for async messaging using this technique and a client test app is available. This is different than using a TAstaClientDataSet. You send out the messages in SendCodedParamList and then you must handle them in the OnCodedParamlist event. Say you send out 5 messages with sql requests. They all go to the server and each will get their own Session from the pool and be launched in a thread using the Messaging thread call ThreadedDBUtilityevent. They will finish executing at different times depending on the size of the result set etc and will be streamed back to the clients OnCodedParamList.

With Power Comes Responsibility

Resource Alert! This means that if your client app starts using this feature and if you fire off 3 requests for async database fetches you will require 3 datamodules on the server to be reserved for the one client. Have a lot of users? do the math!

The AstaBDEServer is coded to accept an SQLStatement in the OnCodedParamList as Msgid 1550.

```
case Msgid of
  1550: AstaServerSocket1.ThreadedDBUtilityEvent(ClientSocket, ttSelect,
  TestUtilSQLEvent, Params);
end;
```

This calls the ThreadedDBUtilityEvent which is used to thread messages and the ttSelect represents the type of database component you require for your event, in this case TestUtilSQLEvent. ASTA delivers you the Query which is what is returned from ThreadedDBSupplyQuery for ttSelect. The result set is stuffed in the ParamList and returned to the client.

Maximum Sessions

You can now define Maximum Sessions for each client. In this case with the BDE Server we started the server with 2 pooled sessions and we don't want it extended. So the MaximumSessions is set to 2 and the MaximumAsyncSessions is put at 2. If a client executes 10 async queries at once, at most only 2 sessions will be used for that client. The rest will be queued up and executed as soon as a session is available.

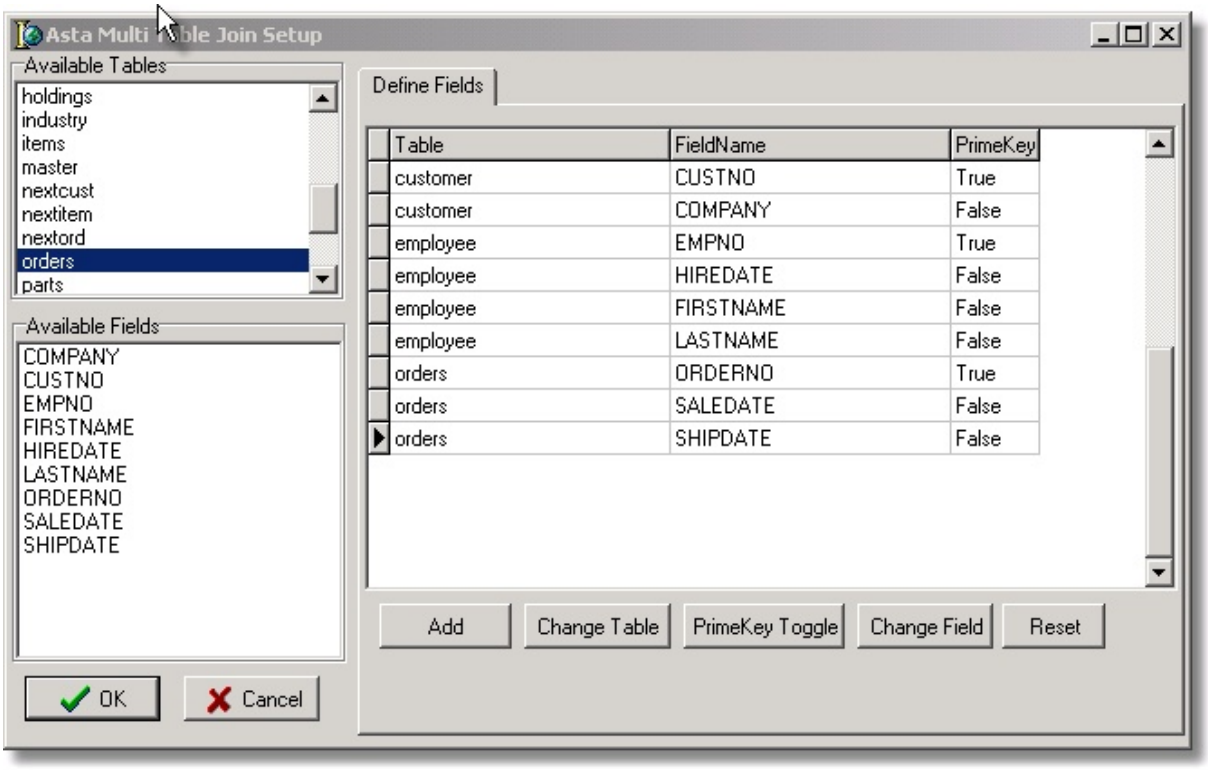
```
AstaServerSocket1.MaximumSessions := 2;
AstaServerSocket1.MaximumAsyncSessions := 2;
```

1.6.2.8 Multi Table Updates from Joins

ASTA supports multiple tables to be updated when using a Cached EditMode. This is not supported in AfterPost edit mode as a transaction is required for the possible multiple lines of SQL generated for any one row.

To use the Multi Table Update Feature:

1. Write your SQL Join and set the AstaClientDataSet.Active property to true at design time
2. Use the EditMode property editor and click on the Multi Table Join Tab
3. Click on the Define fields for multiple table updates button
4. Choose an Update tablename for each field and tag the primekey fields.



Note: the PrimeKeys of all the update tables must be included in the Join as they will be required for the SQL.

1.6.2.9 SQL Generation

ASTA can generate SQL either on the client or on an ASTA server. You may configure ASTA to generate SQL according to your database requirements.There are several properties and events that can be changed to customize your SQL syntax. The following properties and events are available on the client using a TAstaClientSocket and on the server using an TAstaSQLGenerator.

Properties
SQLOptions

UpdateSQLSyntax
SQLDialect

Events
OnCustomParamSyntax
OnCustomSQLSyntax

When SQL is used on the client, SQL for inserts, updates and deletes is generated by ASTA and posted to the server. When there is NO SQL on the client, inserts, updates and deletes are accomplished by sending an OldValueDataSet and a CurrentValueDataSet to TAstaProviders which generate SQL on the server and provide events that you can code to get "in between" this SQL generation.

See also More on SQL Generation.

1.6.2.10 More on SQL Generation

ASTA can generate SQL either on the client or on an ASTA server. On the client side the TAstaClientSocket has the following properties and events to allow you to customize your SQL. If you are using server side SQL generation then you must have an ASTASQLGenerator connected to the TAstaServerSocket on your ASTA server. The AstaSQLGenerator acts similarly to the TAstaClientSocket in allowing you to customize your SQL. SQL dialects can vary greatly between databases and locations. ASTA provides certain properties and events to allow you to customize your SQL to fit your database and your locale.

AstaClientSockets and ASTASqlGenerators contain an SQLOption property of type TAstaServerSQLOptions that is defined in AstaSqlUtils.

```
TAstaServerSQLOptionTypes = (ssUseQuotesInFieldNames,
  ssTableNameInInsertStatments, ssBooleanAsInteger,
  ssUSDatesForLocalSQL,
  ssTerminateWithSemiColon, ssNoQuotesInDates, ssDoubleQuoteStrings,
  ssUseISInNullCompares);
TAstaServerSQLOptions = set of TAstaServerSQLOptionTypes;
```

Value	Meaning
ssUseQuotesInFieldNames	Used for Paradox or any database that allows spaces in field names. Databases like Access need this set to false.
ssTableNameInInsertStatments	
ssBooleanAsInteger	MS SQL Server has bit fields that map to ftboolean in the VCL but then the SQL must be generated as Integers, eg. 0 or 1.
ssUSDatesForLocalSQL	Interbase, Paradox and xBase using the BDE don't like international dates so this forces the server to receive US formatted dates.
ssTerminateWithSemiColon	Access likes its SQL statements to be termintaed with semi-colons.
ssNoQuotesInDates	Use this if you don't want to use single quotes in dates and datetimes.
ssDoubleQuoteStrings	Use this if your strings don't need single quotes but double quotes.
ssUseISInNullCompares	Paradox doesn't like = null and wants IS null.

There may be times where these settings are not enough to customize the SQL that ASTA generates. In those instances you can use the DateMaskforSQL and DateTimeMaskForSQL properties to tell ASTA how to format your ftDate and ftDateTime fields in select statements and update and insert statements.

There are also two events available on the TAstaClientSocket and the TAstaSQLGenerator that allow any field to be formatted to your specifications. Note that you MUST set the boolean parameter Handled to true if you are overriding the default format that ASTA uses. It is up to you to format the string parameter TheResult to your specifictions.

```
procedure TForm1.AstaClientSocket1CustomSQLSyntax(Sender: TObject; DS:
  TDataSet; FieldName: string; var TheResult: string; var Handled:
  Boolean);
begin
  case DS.FieldByName(FieldName).DataType of
    ftDate: begin
              TheResult := QuotedStr(FormatDateTime('DD/MM/YYYY',
  DS.FieldByName(FieldName).AsDateTime)));
              Handled := True;
            end;
  end;
end;
```

The TAstaClientSocket has an UpdateSQLSyntax property. Since there is no standard date and date time SQL syntax, this allows you to set the way date and datetime fields are represented in the SQL generated by ASTA. The default is usBDE and should be used with the BDE Server. usODBC should be used for the supplied ODBC Server. Since Access

has it's own idiosyncrasies, there is also a usAccess choice.

If the SQL that ASTA generates for your database is not correct, you may write a method handler that allows you to supply the correct SQL syntax.

Use the OnCustomSQLSyntax event from the AstaClientSocket. Here is an example of how to create your own SQL override event for an AstaClientSocket.

```
procedure TForm1.AstaClientSocket1OnCustomSQLSyntax(Sender: TObject; DS:
  TDataSet; FieldName: string; var TheResult: string; var Handled:
  Boolean);
const
  SingleQuote = #39;
begin
  case DS.FieldByName(FieldName).DataType of
    ftDateTime,
    ftDate:
      begin
        TheResult :=  SingleQuote + DS.FieldByName(FieldName).AsString +
  SingleQuote;
        Handled := True;
      end;
  end;
end;
```

Remember to set Handled to true so that ASTA knows that you are going to be handling these fields.

1.6.3 ASTA Clients

1.6.3.1 ASTA Jump Start Tips

1. Always code the AstaclientSocket.OnDisconnect event even with a comment.

2. Run an ASTA server unchanged first and write some Client side SQL before you start to code the server so you can get a "feel" for the way things work

3. Don't worry about writing Update, Insert or Delete SQL Statements as ASTA was designed so that you only need write Select SQL and then just set the EditMode property of the AstaClientDataSet.

4. Do not tried to open AstaClientDataSets in the formcreate method. The earliest you can open an AstaClientDataSet is in the OnConnectmethod of the AstaclientSocket.

5. If you want to reconnect an ASTA ClientSocket after it has disconnected, you cannot set the Active to true Property in the OnError or OnDisconnect Event. Use a TTimer instead to allow the event to be passed through and to have the timer trigger the Active := True outisde of the OnError or OnDisconnect event.

6. If you want to force a design time repopulation of an AstaClientDataSet, click on SQL Workbench and Design Populate. This is useful also if you have altered a table and want to force ASTA to go to the server at design time.

7. If you have an AstaClientDataSet open at design time do not try to open it in the OnConnect method. You can do one OR the other but not both.

8. NEVER set the AstaClientSocket.Active at design time. The Borland sockets that ASTA inherits from do not support design time connection.

9. To use the AutoClientUpdate feature you must have your clients log into the server.

1.6.3.2 ASTA SmartWait

ASTA Clients use Asyncronous non-blocking event driven sockets to communicate with remote ASTA servers. This means that typically a request is sent to the server and the client does not "wait" or "block". Any response from the server is received in a event and is thus called Event Driven Programming.

Database Applications are difficult to code using this model since developers need to write procedural code such as:

```
AstaClientDataSet.Open;
while not AstaClientDataSet.Eof do begin
  //do something
end;
```

To allow asyncronous event driven sockets to simulate procedural coding techniques, ASTAClientDataSets trigger ASTA Smart Wait to eat windows messages whenever a Database request is being made to a remote server. ASTA has hidden the asyncronous nature of sockets from the developer to allow for procedural code to be written but there may be times when you want to use asyncronous messaging in your application. ASTA allows this with ASTA Messaging and ServerMethods with the DelayedAction Boolean set to true.

When your client appliction is first executed you will need to understand how the asyncronous sockets work to know when the remote server is actually connected to in order to then access the database.

1.6.3.3 ASTA State and Stateless Clients

TCP/IP maintains state so that when a client connects to a server they stay connected until either client or server wants to force a disconnect, or there is a network error forcing a disconnect.

HTTP is a stateless protocol and runs on top of TCP/IP. It is the protocol that has made the Internet famous and ubiqutious for use by browsers. Browser clients use HTTP to connect to a remote web server, make a request, and then disconnect. HTTP typically runs on port 80.

TCP/IP and State

The original ASTA architecture used TCP/IP as the protocol to communicate between ASTA servers and clients. TCP/IP has a number of benefits which include speed, and the abililty to "push" data from the server which also enables communication between clients.

Features like ProviderBroadcasts allow for any interested client to be notified immediately if another user makes any change to a database via a TAstaProvider. This feature would

not be possible, in real time, if the clients were running stateless.

Since the client is always connected to the server, performance is enhanced as authentication and connection happens only one time. ASTA servers maintain a user list of connected clients. Clients that are stateless are only in this user list for a very short period of time. ASTA does have features to support a stateless userlist.

ASTA supports a login or authentication process where a user connects and gets authenticated and added to the AstaServerSocket.UserList. Of course if clients are stateless, the standard authentication mechanism designed for TCP/IP will not function. ASTA supports this with stateless cookies.

There may be times when TCP/IP is not appropriate or possible. These include when a firewall is present on the client, when using ASTA client components in ISAPI dll's, or in handheld development.

Firewalls
http://www.astatech.com/support/white/firewalls.asp

Many firewalls do not allow allow TCP/IP traffic but only HTTP traffic via browsers. ASTA supports this through HTTP tunneling where the ASTA messages are "wrapped" or disgusised as HTTP. In order to use HTTP tunneling with ASTA, a web server is required. An ISAPI dll is available in the ASTA installation named Astahttp.dll. This dll is copied to a web server like IIS and installed in a directory that allows for ISAPI dll's to be run. Typcially in the scripts directory for IIS.

The AstaClientSocket has a WebServer property that allows for the configuratio of the AstaClientSocket to use http either using ASTA sockets or using the Microsoft DLL WinInet.dll. WinInet.dll comes with any version of IE (Internet Explorer). Using the ASTA WinInet option is the recommended way to traverse any firewall as it uses settings already in place by Internet Explorer.

Clients that run HTTP connect and disconnect to the server on each request. There is overhead in this. Additionally, there is no userlist on the AstaServer socket when running stateless and the normal ASTA authentication process does not apply. ASTA supports the authentication of stateless clients. For a discussion of this see the Cookies Discussion.

Blocking Clients
ASTA sockets are async and event driven and do not block or require threads. They use the recommended mode of event driven socket programming by Microsoft using windows messages. Windows programs contain a "message" pump to handle windows messaging. If you are coding an ISAPI dll there is no message pump so ASTA clients must run in blocking mode. See the Blocking Discussion for more detail on this.

1.6.3.4 Cached Updates

Asta supports data-aware controls and can generate SQL to automatically transmit update, insert and delete statements to the ASTA server. To enable automatic SQL generation you must set three public properties using the EditMode property editor. Make sure you set some SQL and make your table active first so that there are actual field names available for ASTA to work with.

The EditMode property editor allows you to set up a TAstaClientDataSet so that ASTA can automatically generate the SQL for you. The EditMode property editor requires that the TAstaClientDataSet have field information already defined by writing an SQL select statement and setting active to true. After using the property editor, it will display text regarding the EditMode status as Read Only, Cached or After Post. For fields that you do not want included in any insert, update or delete statement set the ReadOnly property of that field to true. For AutoIncrement fields, just pull down the AutoIncrement property with that field name and it will not be used on insert statements. of ASTA can Refetch the auto increment value of a field, and other fields automatically after an insert statement, as well as after update statements if the AstaServer you are running supports this.

UpdateTablename	This is the name of the table that is to be updated and works just like the TableName property with a pull down list of tables available on the server.
PrimeFields	Asta needs to know what fields are used to determine uniqueness.
Update Method	If the database supports transactions (ASTA adds a begin transaction/commit pair to SQL statements) you may set this property to umCached. Asta will track any table inserts, edits or deletes and you must manually call the ApplyUpdates method. This will send the SQL to the server. If you set this property to umUAfterPost then on each row change the SQL will be sent to the server. The default is umManual.

```
function ApplyUpdates(TransactionMethod: TUpdateSQLMethod): Boolean;
TUpdateSQLMethod = (usmNoTransactions, usmUseSQL, usmServerTransaction,
  usmCommitAnySuccess);
```

ApplyUpdates can be handled in one of four ways:

No Transactions	This is what is called if the EditMode is set to umAfterPost and updates one row at a time.
UsmUseSQL	This applies cached updates and uses the TAstaClientSocket.SQLTransactionStart and SQLTransactionEnd to post the SQL to the server. The default for SQLTransactionStart is BEGIN TRANSACTION and the default for SQLTransactionEnd is COMMIT. ODBC for instance, by default is set to auto commit mode, and by using these paired statements surrounding multiple lines of SQL, transactions can be invoked.
UsmServerTransaction	This posts the SQL to the server and calls the explicit transaction support that ASTA servers have. The AstaBDEServer will use the TDataBase.StartTransaction call. The ASTAODBCExpress server will use the Thdbc.StartTransact. See the TAstaTransaction.doc for a discussion on transactions.
UsmCommitAnySuccess	This posts the SQL to the server just like usmServerTransaction however a Commit will be called on the server if any SQL sent to the server is successful.

To set the Editmode at run time use the method:

```
procedure SetEditMode(const Primes: array of string; TheUpdateTable:
  string; TheUpdateMethod: TAstaUpdateMethod);
```

Here is an example from the DataAwareRunTime tutorial showing how a TAstaClientDataSet can be changed from editing a Customer Table to editing an Employee Table:

```
procedure TForm1.BitBtn1Click(Sender: TObject);
begin
  with AstaClientDataSet1 do begin
    Close;
    SQL.Clear;
    SQL.Add('Select * from Employee');
    Open;
    SetForUpdate(['EmpNO'], 'Employee', umAfterPost);
  end;
end;
```

1.6.3.5 Parameterized Queries in a Transaction

To send plain update, insert and delete ExecSQL in a transaction you can use SendSQLTransaction or SendSQLStringTransaction. If you want to send a list of Parameterized ExecSQL statements you can use the following methods.

```
procedure ClearParamterizedQuery;
procedure AddParameterizedQuery(TheSQL: string; P: TAstaParamList);
function SendParamterizedQueries: TAstaParamList;
function ParamQueryCount: Integer;
```

Example

```
var
  p, p1: TAstaParamList;
  i: Integer;
begin
  p := TAstaParamList.Create;
  p.FastAdd('LastName', 'Borland');
  p.FastAdd('EmpNo', 2);
  with AstaclientDataSet1 do begin
    ClearParameterizedQuery;
    AddParameterizedQuery('Update employee set LastName = :LastName
  where EmpNo = :empNo', p);
    p.parambyName('Empno').AsInteger := 4;
    AddParameterizedQuery('Update employee set LastName = :LastName
  where EmpNo = :empNo', p);
    p.Clear;
    p.FastAdd('Phone', NewExt.Text);
    p.FastAdd('PhoneExt', OldExt.Text);
    AddParameterizedQuery('Update employee set PhoneExt = :Phone where
  PhoneExt < :PhoneExt', p);
    p1 := SendParamterizedQueries;
  end;
    Memo1.Lines.Clear;
    //params will include the orginal SQL as the name and either an
```

```
    Error message or the
      //rows affected in the actual param
      for i := 0 to p1.count - 1 do
        memo1.lines.add(p1[i].Name + ':' + p1[i].AsString);
     p.free;
     p1.free;
end;
```

1.6.3.6 Refetching Data on Inserts and Updates

ASTAClientDataSets can refetch values from the server on inserts and updates. Use your down arrow in this grid to add a row or two but only put something in the SomeField column. Leave the id and default field fields alone. For master/detail there is also autoncrement support in the SendMasterDetailAutoIncTransaction method of the AstaClientSocket.

1. Set up your select query and set your edit mode to cached. Refetch on insert action can only work within a transaction so your database must support transactions.

2. Set the AutoIncrementField Property

3. Set the fields that you want to refetch from the RefetchFieldsOnInsert Property

4. Run your project and call apply updates after you have inserted a row or 2. In this demo the auto increment value and the default field value are streamed back to the client with NO code needed by you!

What do you need to do? Not much. Just code one event on the AstaServerSocket.

```
procedure TForm1.AstaServerSocket1InsertAutoIncrementFetch(Sender:
TObject; ClientSocket: TCustomWinSocket; AQuery: TComponent;
TheDatabase, TheTable, TheAutoIncField: string; var AutoIncrementValue:
Integer);
begin
  //put your code in here.
  //Something like
  with aQuery as TQuery do begin
    SQL.Add('SELECT max(' + TheAutoIncField+') FROM ' + TheTable);
    Open;
    TheAutoIncrementValue := FieldByName(TheAutoIncField).AsInteger;
  end;
end;
```

The above code is specific to your database. For access, paradox and Interbase 5.X the above code should work in calling MAX to get the last value used by the server for an autoinc field. For MS SQL Server, Sybase, and SQL Anywhere you would need to write code to access the @@identify variable. The AstaODBC server is already coded to identify these databases and make the appropriate call. For Oracle you would need to get the last sequence value.

Here's what happens on the server:
1. an insert statement comes in either alone or within a transaction

2. it is executed then then the AstaServer socket calls the OnInsertAutoIncrementFetch which
returns a Var Integer of the autoinc value just used for the insert statement. of course

the server must support transactions to make this work as the value returned is only valid
to your session in a transaction.

3. If the autoinc is the only field requested to be refetched it is streamed back to the asta
clientdatset if the transaction is successful. If you have tagged other values in the AstaClientDatSet.RefetchOnInsert property, an additional select statement is fired that retrieves
any fields that you have choosen (default server values, values changed by triggers etc) and all the values are streamed back.

1.6.3.7 Streaming Asta Datasets

TAstaDataSets are streamable so that they can be saved to disk or even stored in the blob field of a database. There are several methods available for streaming:

```
procedure SaveToFile(const FileName: string);
procedure LoadFromFile(const FileName: string);
procedure SaveToStream(Stream: TStream);
procedure LoadFromStream(Stream: TStream);
function SaveToString: string;
procedure LoadFromString(S: string);
procedure LoadFromXML(const FileName: string);overload;
procedure LoadFromXML(Const Stream:Tstream);overload;
procedure SaveToXML(const FileName:string;XMLFormat :
TAstaXMLDataSetFormat);overload;
procedure
SaveToXML(Stream:TStream;XMLFormat:TAstaXMLDataSetFormat);overload;
```

When these methods are called both the defined fields and the data is streamed. Streaming memos and blobs is fully supported. Since the field definitions are stored in the form (.dfm) file, TAstaDataSets do NOT load their field definitions when LoadFromStream or LoadFromFile is called. This is done so as not to interfere with any persistent fields that you have defined. The fields are, however, always stored in the saved streams. To load a TAstaDataSet that has been saved from another TAstaDataSet that has no fields defined, use the following functions:

```
procedure LoadFromStreamwithFields(Stream: TStream);
procedure LoadFromFileWithFields(const FileName: string);
```

This allows datasets of datasets to be saved if a blob field is defined in a TDataSet.

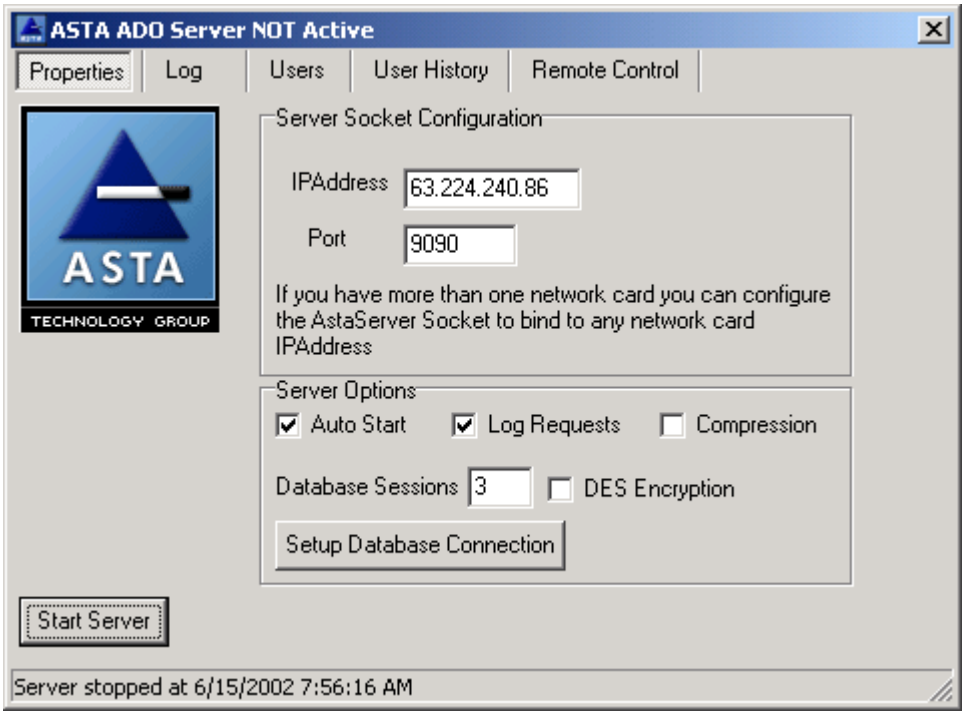
1.6.4 ASTA Servers

ASTA servers come out of the box with the ability to

- Run Threaded
- Support Client Side SQL
- Remote Adminstration
- Remote control
- Run as a normal EXE or NT Service

- Support the ASTA ODBC Driver
- Support SkyWire (non-vcl) Clients like Palm and WinCE
- Support JDBC Driver

Below is a screen shot of the Asta3ADOServer. ASTA 3 servers use a specific format.



1.6.4.1 AstaServerFormat

ASTA Servers can run as NT Servicers or normal EXES. The following is a discussion of the standard format of ASTA servers. This discussion will use the Asta3ADOServer as an example but the same concepts apply to all ASTA servers as they are converted to the new format. ASTA 3 ships with 10+ servers in the new server format with 30+ Servers available using Delphi 3rd Party Database Components. If you do not see a server using the Delphi 3rd Party Database Component you want to use contact ASTA.

The Asta3ADOServer consists of source files:

1. Asta3ADOServer.dpr
2. AstaADOSupplement.pas
3. MainForm.pas which uses AstaServiceUtils.pas
4. SocketDM.pas
5. DM.pas

The MainForm.pas has the GUI and options to configure the server Port, compression, encryption etc and implements a form that descends from TAstaServiceForm as defined in AstaServiceUtils.pas. This has a lot of housekeeping internals to allow the server to run

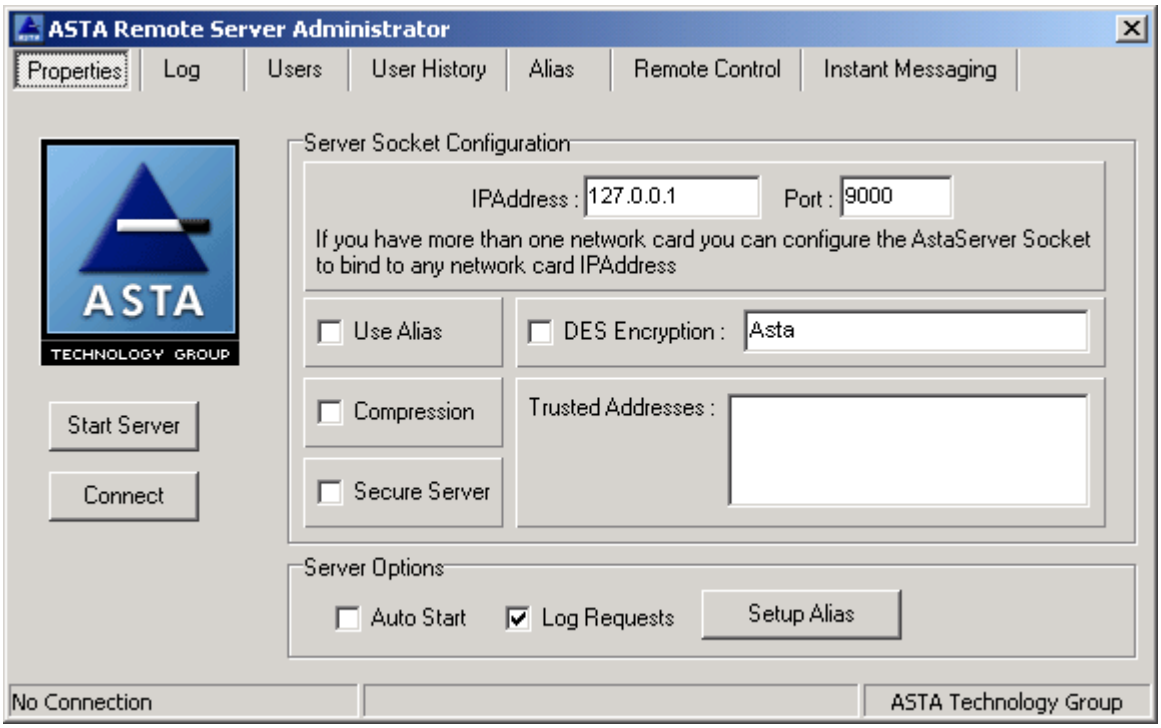
as a normal EXE or NT Service.

The DM.pas file is where the Database Components reside. If you want to put the "Business Logic" on the server using TAstaProviders to ASTA Server Methods, code it on this form. By coding it on the DM.pas, and not accessing any database components beyond that form, your Database code will be thread safe and ASTA will handle the threading for you. You can also add as many other DataModules as you desire. Any Messaging should be coded on the SocketDM.pas unit and that is where you can also add your PDA Support, with one like of code if you don't want to customize your server for Palm or WinCE clients but use the SkyWire API only.

Database Support
Remote Admin
Messaging
NT Service
Remote Control
NT Service

1.6.4.1.1 AstaServerFormat.RemoteAdmin

ASTA servers can be administered from a remote Admin Client. Below is a screen shot of the AstaServerAdmin that comes with the ASTA Server Download. The AstaServerSocket.ServerAdmin allows you to set up the server to be administered by are remote admin client. Internally the AstaServerSocket instantiates another AstaServerSocket on another port for security purposes.



1.6.4.1.2 AstaServerFormat.RemoteControl

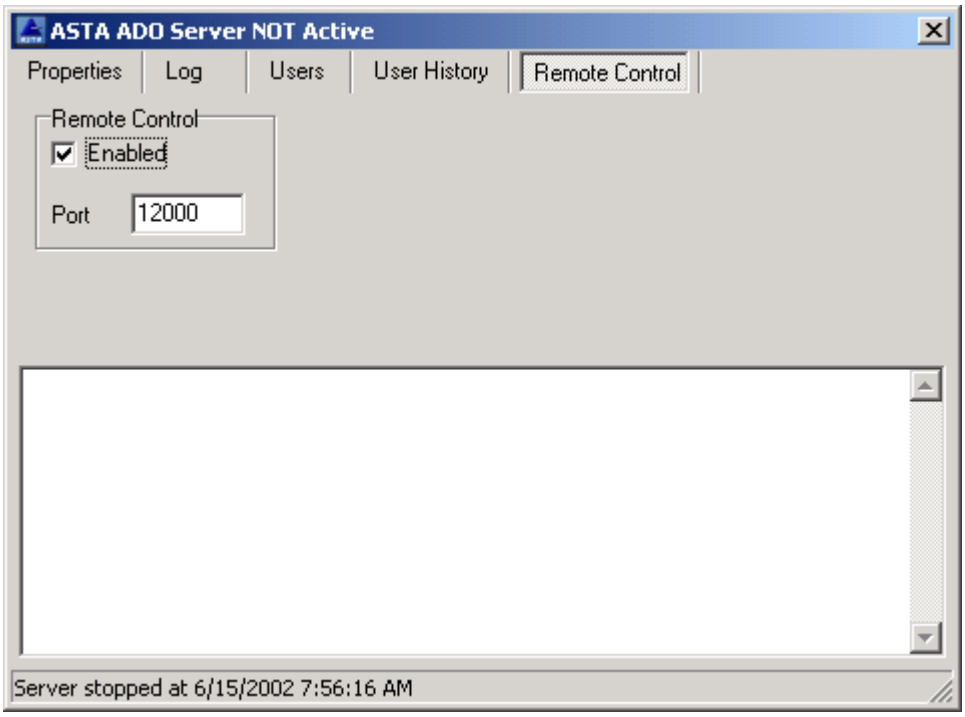
ASTA Servers support the concept of RemoteControl, just like PcAnywhere where the machine that a server is running on can actually be fully controlled by a remote ASTA client. To enable this feature, ASTA servers have a table where the RemoteControl client port can be configured and the remote control client can be enabled.

If you want to enable this feature on your ASTA server make sure the compiler directive

```
{$DEFINE AstaRemotecontrol}
```

is defined in AstaServiceUtils.pas and that your project adds the path to the AstaRemote Control Components (\Asta\RemoteControl) to your server.

This feature can also be used in your ASTA client applications. You can use an AstaServerSocket to support your users. ASTA servers licenses do not apply in this case as ASTA allows you to use the AstaServerSocket and RemoteControl component (found in your ASTA\RemoteControl directory) to then connect to your clients running your ASTA app and actually take over their keyboard.



1.6.4.1.3 AstaServerFormat.CodingTheServer

To extend your ASTA server with Messaging, code the SocketDM.pas unit. This is the the unit that has the AstaServerSocket on it. To extend the server with TAstaProviders and ServerMethods using the TAstaBusinessObjectsManager code the DM.pas unit which contains the Delphi Database Components used on your ASTA server. ASTA also supports any number of other DataModules to be used in your server. We have heard of one user coding 100 DataModules on an ASTA server!

1.6.4.1.4 AstaServerFormat.Messaging

When coding your ASTA server using messaging calls use the SocketDM.Pas unit where the AstaServerSocket is found.

Here is an example of the TAstaServerSocket.OnOutCodedParamList event from the Asta3ADoServer SocketDM.pas unit.

```
procedure TServerDM.ServerSocketOutCodedParamList(Sender: TObject;
  ClientSocket: TCustomWinSocket; MsgID: Integer; InParams,
  OutParams: TAstaParamList);
begin
  outParams.FastAdd('ServerTime',now);
end;
```

1.6.4.1.5 AstaServerFormat.DatabaseSupport

ASTA Servers support something called Client Side SQL and in order to do this there are certain features that are required to be supported like supporting SQL Selects, Parameterized Queries and Transactions. Since each Delphi 3rd Party Database component uses different properties and mehtods, ASTA has abstracted the whole Database process in the unit AstaDatabasePlugin.pas.

Each ASTA 3 server, using the new Database format will have a unit called AstaXXXDatabaseSupplement.pas that implements a descendent of the TAstaDatabasePlugin.

in the Asta3ADOServer it is implemented in the Asta3ADOSupplement.pas file which declares a

```
TAstaADOPlugin = class(TAstaDataBasePlugin)
```

Source to this file comes with the ASTA server you are using.

1.6.4.1.6 AstaServerFormat.NTService

Asta Servers can be run as Services or stand alone EXE's. To run as a service, run the server one time to setup the Server port and other options and then run type from the command line

Asta3ADOServer /install

and it will be installed as an NT Service

Below is the code from the Asta3ADOServer.dpr project file. ASTA uses the registry to store information about the settings you set from your server. If you want to run multiple copies of an ASTA server as an NT Service just copy the EXE to another file name like AstaNorthwindServer.exe and that server will have it's own set of registry settings and run as an NT Service completely separate from the Asta3ADOServer

```
begin
  //AstaServiceName:=AstaDefaultServiceName;
  //default will be Asta3ADOServer
  //if you need to run 2 instances just rename the Exe
  AstaAppInstancecheck;
  if Installing or StartAstaService then
  begin
    SvcMgr.Application.Initialize;
    SocketService := TAstaService.CreateNew(SvcMgr.Application, 0);
    // use this next constructor if you have any dependencies where AServiceName is the name
of the service
    //SocketService.TAstaService.CreateNewDependency(SvcMgr.Application, 0. AServiceName:
String);
    Application.CreateForm(TAstaServiceForm, AstaServiceForm);
  Application.CreateForm(TServerDM, ServerDM);
  SvcMgr.Application.Run;
  end else
  begin
    Forms.Application.ShowMainForm := False;
    Forms.Application.Initialize;
    Forms.Application.CreateForm(TAstaADOServiceForm, AstaServiceForm);
    AstaServiceForm.Initialize(False);
    Forms.Application.Run;
```

```
end ;
end .
```

Dependencies

If you are running your ASTA server on the same machine as your database you may want to configure the dependencies to wait for another service to start. Below is some code examples of how to do this for the Interbase Server and the Interbase guardian.

```
// call this constructor if you need to add a dependency
// you add any dependencies here using IB server to start up on reboot.
// NOTE: make sure you separate each name with a '|', and add '|'  at end of string too!!!
    SocketService := TAstaService.CreateNewDependency(SvcMgr.Application, 0,
'InterBaseServer|InterbaseGuardian|');
// call this next constructor if there are no dependencies to worry about
```

1.6.4.2 ASTA Database Server Support

ASTA servers are available using the following Delphi components:

Adonis	
Advantage Database Engine	
Asta3ADOServer	ADO support
Asta3BDEServer	BDE support
Asta3DBISAMServer	DBISAM support
Asta3DOAServer	DOA support
Asta3IBExpressServer	IB Express
AstaCTLibServer	Native access to Sybase and Adaptive Server Anywhere
AstaOdacServer	
AstaSkywireEmulator	
Diamond Access	
Direct Oracle Access	
Flash Filer	Coming soon - www.turbopower.com
Halcyon	DBF server – www.grifsolu.com
InterBase Objects	
MySQL	Using WinZeos components
NativeDB	Native SQL Anywhere
NCOCI8	Native freeware Oracle 8i server –
ODBC98	
ODBCExpress	
Opus Direct	Provides BDE replacement, use the AstaBDEServer – www.opus.ch
Oracle Direct Access	Core Labs (under development)
SQL Direct	MS SQL Server, Oracle, Informix, ODBC
SQLQuery	TheComponentStore

For more detailed information visit: http://www.astatech.com/support/servers.htm

1.6.4.3 ASTA Middleware Triggers

To further extend ASTA servers as the most appropriate vehicle for the ASP Model (Application Service Provider) ASTA introduces ASTA Middleware Triggers.

A large reason to use N Tier architecture is to allow for Database Connections to be shared between remote clients and leverage this important resource. ASTA's pooled Sessions threading model allows for ASTA servers to scale as they "obtain resources late and release them early".

Traditional 2 Tier database applications techniques rely on Database Triggers to fire on when update, insert and delete SQL is executed against the Database. Using Pooled Database connections along with an Application Server like ASTA, does not allow for each user to have their own connection to the database.

ASTA Middlware Triggers allow ASTA users to "register" TableName.FieldName for applications and to supply a default value for that Field that is maintained on ASTA servers and to also fire an event on the server for every SQL statement executed in a transaction.

To use AstaTriggers on the client you must register the Tablename.Fieldname that you want default values to be fired against on the server.

The following shows an example of registering the Customer.UserID:Integer field as an ASTA trigger.

```
AstaClientSocket1.RegisterTrigger('Customer', 'UserID');
```

A default value must also be supplied. This can be done from the client application or within the Login process on the AstaServer. From the Client you would call:

```
ClientSocketParams.FastAdd('UserID', 1002);
```

From the server you would use:

```
AstaServerSocket1.UserList.GetUserParamList(Sender as
  TCustomWinSocket).FastAdd('UserID', 1002);
```

Note the UserID of 1002 supplied depending on the Client UserName and Password.

There is a new event on the AstaServerSocket.OnTriggerDefaultValue to support supplying the values for any triggers as well as an OnTriggerEvent that gets fired for any SQL executed from a client initiated ApplyUpdates call..

```
procedure TForm1.AstaServerSocket1FireTrigger(Sender: TObject; U:
  UserRecord; SQLString, Tablename, FieldName: string; P:
  TAstaParamList);
begin
  if (compareText(tablename,'Customer') = 0) and
  (comparetext(fieldname,'USeRID') = 0) then begin
    logit(' server trigger ' + SQLString);
    p.FastAdd('UserID', u.FParamList.ParamByName('USerID').AsInteger);
```

```
    end;
  end;
```

The result is that if the Customer row had a userId field that contains the UserID of the last user that updated the row, a Parameterized Query is generated and the UserID param is supplied with values that exist on the server based on the ClientSocket:TCustomWinSocket, not the client.

Start Transaction 63.224.240.84 Transaction
Server trigger UPDATE customer SET customer.Company ='Borland',customer.UserID= :UserID
Where customer.CustNo = 1231
Param 0->1002
Transaction Commited

1.6.4.4 ASTA Server Admin

ASTA 3 servers now have the concept of a ServerAdmin. There is an subproperty of the AstaServerSocket named ServerAdmin. This allows for the AstaServerAdmin.exe to be run and connect to an internal AstaserverSocket instatiated and connected solely to be used for remote administration.

```
 TServerAdmin = class(TPersistent)
  published
    property LogOptions: TAdminLogOptions read FLogOptions write
  FLogOptions
default [loShowLog, loLogins, loDisconnects];
    property FeaturesSupported: TAdminFeatures read FFeaturesSupported
  write
FFeaturesSupported;
    property Active: Boolean read FActive write FActive default False;
    property Address: string read FAddress write FAddress;
    property Port: Word read FPort write FPort;
    property UserName: string read FUserName write FUserName;
    property PassWord: string read FPassword write Fpassword;
    property Encryption: TAstaEncryption read FEncryption write
  FEncryption;
    property TrustedAddresses: TStrings read GetTrustedAddresses write
SetTrustedAddresses;
  end;
```

If the AstaServerAdmin.Active property is set to True, then an AstaServerSocket descendent (a TAstaServerAdminSocket defined in AstaServerAdmin.pas) is created listening at the Address and Port as defined in the AstaServerSocket.ServerAdmin sub property. A UserName and Password as well as Encryption method can be defined. Addresses defined as "Trusted" can be added and will authenticate regardless of username/password.

There is a \Asta\ServerAdmin directory that contains a AstaServerAdminClient.dpr that can connect to the AstaServerSocket.ServerAdmin. This client can optionally receive the logs normally shown on ASTA servers. This opens up the next step which will be to optionally have ASTA servers run hidden with no tray icon.

AdminClients can show connects, disconnects, ServerExceptions (not implemented yet) and also show the UserDataSet and UserHistoryDataSet used on ASTA 3 servers. Some ASTAServers, like Advantage and Dbisam allow the concept of "aliases" where multiple

databases/directories can be supported.

The FeaturesSupported property of the TServerAdmin allows additional features to be defined so that the ServerAdminClient can support extended features.

```
TAdminFeature = afAlias, afAutoUpdates, afTrustedAddresses, afUserLogin,
  afEncryption, afCompression, afRemoteControl, afCustom);
```

1.6.4.5 ASTA Server Logging

ASTA servers, by default come with logging turned on but with an option to turn it off. By definition, logging is an expensive activity and is not thread safe. Writing to a UI component (TMemo) is not thread safe and eats a lot of CPU cycles.

ASTA supports logging by coding the AstaServerSocket OnShowServerMessage event. Internally the RecordServerActivity Method calls OnShowServerMessage. There is a Logon:Boolean property that allows for all logging to be turned on or off and ASTA 3 servers allow this to be controlled by the Server UI or with the RemoteAdmin feature.

If you require more granualar control of the logging process, you can define which events you want the server to log.

```
TAstaLogItem = (alLogAll, alThread, alException, alProvider,
  alServerMethod, alProviderBroadCast, alRefetch, lLogin,
  alClientUpdate, alPersistentSession, alJava, alGeneral, alDisconnect,
  alHttp, alUtilityThread, alCreateSession, alDisposeOfSession,
  alSendExceptionToClient, alThreadError, alSelectSQL, alExecSQL,
  alMetaData, alStoredProcs, lFailedLogins, alPdas, alSessionCheckOut);

TAstaLogItems = set of TAstaLogItem;
TServerLogEvent = procedure (Sender: TObject; LogItems: TAstaLogItems;
  ClientSocket: TCustomWinSocket; Msg: string; var RecordOnMainLog:
  Boolean) of object;

property NoLogItems: TAstaLogItems;
property LogItems: TAstaLogItems;
property LogOn: Boolean;
```

You can specify that you want to log all items and add items that you don't want to log to the NoLogItems property, or you can define the actual items you want logged.

```
property LogEvent: TServerLogEvent;
TServerLogEvent = procedure (Sender: TObject; LogItems: TAstaLogItems;
  ClientSocket: TCustomWinSocket; Msg: string; var RecordOnMainLog:
  Boolean) of object;
```

Internally ASTA calls:

```
procedure LogServerActivity(LogItems: TAstaLogItems; S:
  TcustomWinSocket; const Msg: string);
```

Which checks the LoginOn, LogItems and NoLogItems. If the LogEvent is coded you can

further control if an item is passed through to the log. Only if all the tests passed is RecordServerActivity actually called.

Note: Problems on ASTA servers are often caused by ASTA users writing logging routines that are not thread safe. Beware of too much logging!

1.6.4.6 ASTA Servers and Multiple DataSources

The ADO Server allows multiple ADOConnections to be created on the fly so that AstaClientDatasets can use the Database:string property to access any number of Databases on the server. Other ASTA servers have been coded to allow for multiple Datasources including the AstaOdbcServer, AstaDbisamServer and AstaBDEServer. There is also a multi-datasource ASTA ISQL Demo tutorial that demonstrates this feature.

How to Define the Available Databases on the Server
ASTAClientDataSets have a Database:string property that can be populated from ASTA servers so that there is a design time pulldown of Database choices. To do this the AstaServer must code the OnFetchMetaData Event to return a TAstaDataSet in response to an mdDBMSName Metarequest from AstaclientDatasets. The server must append the DbmsDataSet and define a value in the Database field that will stream back to the remote clientdataset.

This ADO Server contains a new menu option of Manager Aliases to allow Alias information to be defined and saved to file by a TAstaDataSet calling SaveToFile. The SetupDBMSDataSet call on the server populates the DBMSDataSet.

```
procedure TForm1.SetupDBMSDataBase;
var
  i: Integer;
  d: TAstaDataSet;
begin
  with AstaDataModule do begin
    with DBMSDataSet do begin
      if FileExists(AliasDataSetFileName) then begin
        D := TAstaDataSet.Create(nil);
        try
          d.LoadFromFilewithFields(AliasDataSetFileName);
          //add the main one you login on or the one you set the
  ADOConnection to on the datamodule
          AppendRecord([AstaServerSocket1.AstaServerName, 'Main',
  ExtractFileName(Application.EXEName)]);
          while not d.Eof do begin
            AppendRecord([AstaServerSocket1.AstaServerName,
  d.FieldByName('Alias').AsString, '',
  ExtractFileName(Application.EXEName)]);
            d.Next;
          end;
        finally
          d.Free;
        end;
      end
      else AppendRecord([AstaServerSocket1.AstaServerName,
  AstaServerSocket1.DataBaseName, '',
  ExtractFileName(Application.EXEName)]);
    end;
    UpdateAstaAlias(DBMSDataSet);
```

```
  end;
end;
```

How to have server side database calls react to the Database:string property coming from the client

AstaClientdataSets will send their Database:string property to ASTA Servers on every database request. the Server must be able to respond to these requests by using or creating ADOConnections that are then connected to the ADO Database components sitting on the server AstaDataModule.

An FDatabaseList is used on the server to store the Alias Names and the AdoConnections that are created on the server in response to a client request for a particular Database.

This works in all 3 ASTA threading models and will create a list of Aliases with TAdoConnection components stored in the FDatabaseList.Objects property. When first adding a TAdoConnection that does not exist there may be a slight delay as the AdoConnection connects to the server, depending on which database you are using but after that it will be very fast and efficient.

Note: When you define an Alias, build a Connection string and paste it into the memo field on the AliasManager setup form. this will be used to connect to the AdoConnection created on the fly.

```
function TAstaDataModule.GetADOConnection(ClientSocket:
TCustomWinSocket; Alias, FileName: string): TComponent;
var
  spot: Integer;
  Ad: TAdoConnection;
begin
  Result := ADOConnection;
  if Alias = '' then Exit;
  //add a line here if you want to skip this test for your default
DataSource
  spot := FDatabaseList.Indexof(Alias);
  if spot < 0 then begin
    if FDataSet.RecordCount = 0 then
FDataSet.LoadFromFilewithFields(FileName);
    if not FDataSet.Locate('Alias', Alias, [locaseinsensitive]) then
Exit;
    if CompareText(AdoConnection.ConnectionString,
FDataSet.FieldByName(AstaAliasconnectionString).AsString) = 0 then Exit;
    Ad := TAdoConnection.Create(Self);
    Ad.LoginPrompt := False;
    Ad.Provider := AdoConnection.Provider;
    Ad.IsolationLevel := AdoConnection.IsolationLevel;
    Ad.CursorLocation := AdoConnection.CursorLocation;
    Ad.ConnectOptions := AdoConnection.ConnectOptions;
    Ad.ConnectionString :=
FDataSet.FieldByName(AstaAliasconnectionString).AsString;
    FDatabaseList.AddObject(Alias, Ad);
    Result := Ad;
    Ad.Connected := True;
  end
  else Result := TComponent(FDataBaseList.Objects[spot]);
```

```
end;
```

1.6.4.7 ASTA Servers and Visual Interfaces

ASTA servers come coded with a logging memo so that server activity is easy to view during the design phase of your ASTA application. There is a tremendous performance cost to writing to any visual control in addition to the fact that Delphi Visual Components are not thread safe. ASTA 2.5 server socket's have a new property of LogOn (type Boolean) that turns on and off writing to the logging memo found on ASTA servers.

The ASTA server socket has an event called OnShowServerMessage which is typically coded like this:

```
procedure TfrmMain.AstaServerSocket1ShowServerMessage(Sender: TObject;
  Msg: string);
begin
  mrequests.lines.add(msg);
end;
```

The AstaServerSocket has a method used internally by ASTA to show threading activity and other internal ASTA server activities for debugging and demonstration purposes, to prove the servers really do thread <g>. RecordServerActivity will call the OnShowServerMessage.

```
procedure RecordServerActivity(S: TCustomWinSocket; Msg: string);
```

The LogOn property just unassigns the OnShowServerMessage event. No event, no log. no cpu degradation. Writing to the log is very, very costly in terms of cpu usage so we recommend that servers have no visual interface.

What ASTA needs to do is to supply a ClientAdmin ASTA app to show server activity. In developing the ASTA Anchor Server which does load balancing and fail over, ASTA added a messaging layer that can return ASTA server information.

1.6.4.8 Building ASTA Servers

ASTA was designed to allow any Delphi database component to be used on ASTA servers and be threaded by ASTA internally to support ASTA's 3 threading models (singlesession, persistentsession and pooled).

ASTA Servers need to be able to handle:

1. Selects
2. Parameterized Queries
3. MetaData Requests
4. Fetch Blob Requests
5. Transactions
6. Stored Procedures

To support the above 6 items, ASTA Servers need to know which components to use for which operation. To that end the ThreadedDBSupplyQuery is called.

ThreadedDBSupplyQuery passes in an enumerated type of TThreadDBAction which can be ttSelect, ttExec, ttStoredProc, ttTransactionStart, etc.

So in your case you are using TAstaProviders on the server to generate SQL on the server from remote AstaClientDataSets.

When you are using an AstaClientDataSet with NO SQL and you want to be able to make it editable and post changes on the server using a provider, you set the SQLGenerateLocation property of the AstaClientDataSet to gsServer. This sets EditMode to Cached and allows you to then call ApplyUpdates(usmServerTransaction) to post any inserts, updates and deletes on the server.

When you call ApplyUpdates, the AstaClientDataSet sends two datasets to the server. The OriginalValues (nothing for inserts) and CurrentValues(nothing for deletes).

The TAstaProvider then looks to (or creates if none exists) a TAstaSQLGenerator to get any SQLDialect settings (for instace MS SQL Server needs booleans to be posted as integers).

1. ASTA calls ThreadedDBSupplyQuery and says "What component do I need to start a transaction?" (ttTransactionStart is passed)
2. In your case your ThreadedDBSupplyQuery will return a TDatabase.
3. ASTA then calls the OnTransactionBegin event of the TAstaServerSocket and passes it the TDatabase it just got from #2, and the transaction is started.
4. We need to execute a bunch of parameterzied queries for the SQL to be generated so ASTA calls the ThreadedDBSupplyQuery with ttExec and your BDE server returns a TQuery.
5. ASTA generates SQL (and with a provider you get a chance to modfiy the values of the two datasets passed from the client or to generate your own SQL in events) for each row of the AstaClientDataset and calls the AstaServerSocket.OnExecSQLParamList using the TQuery that it has.
6. This repeats for all the rows (and refetches can be gathered up for things like auto increment fields that can be streamed back on inserts).
7. At the end of the transaction ASTA calls the OnTransactionEnd passing the TDatabase again, and either commits or rolls back the transaction according to what has happened on the executes.

1.6.4.9 Coding ASTA Servers

ASTA Servers were designed to be able to support any kind of Database component that can be used with Delphi. Currently ASTA supports over 30 Delphi Database Components on ASTA Servers. One of the design goals of ASTA was to handle the threading issues internally so that the Application developer would never have to worry about such low level details.

SQL Selects
Parameterized Queries
Metadata Requests
Fetch Blob Requests
Transactions
Stored Procedures

To thread ASTA servers there are some additional events that must be coded.

SupplyDatabaseWorkerComponents

SupplyDatabaseSessions

ASTA Servers are threaded internally by ASTA so there is never a need for you to code any threading. If you code the required events your ASTA server will be able to run in all 4 ASTA threading models.

ASTA Server Format
The Big Picture

1.6.4.10 Command Line Switches

The following command line switches are available for the AstaBDEServer and AstaODBCServer. See also AstaServerSocket.CommandLinePortCheck.

Note: The new ASTA 3 Server format deprecates old command line switches used in ASTA 2.6.

Compression	Sets the Compression property to acAstaCompress.
Port	Sets the Port property
Login	Sets the AstaClientSocket.AutoLoginDlg to ltLoginDlg
NoDesignTime	Turns off Design Time Access to an ASTA server by setting the DTAccess property to false. A DTUserName and DTPassword is required on the server to use this feature.
NoLog	Turns off the AstaServerSocket.Logon property

Examples
AstaBDEServer.exe Port=9012
AstaBDEServer.exe Port=8080
AstaODBCServer.exe Port=1111 Compression

1.6.4.11 Data Modules on ASTA Servers

To insure that ASTA servers thread properly, ASTA server side components must be deployed on a DataModule on the server. This allows for ASTA to create copies of the DataModule so that each user can have their own unique copy of it for database work within a thread. This includes any Delphi 3rd party database components that you are using and Asta components with the exception of TAstaServerSocket and TAstaSQLGenerator which should be kept on the mainform of the application as only one instance of each of these components are required for the server.

To use ASTA's server side components (ServerDataSets, TAstaProviders, TAstaBusinessObjectManagers) you must make a call to RegisterDataModule in the AstaServers FormCreate event. This allows ASTA to take an internal inventory of server

side components so that remote clients can be made aware of what is available on the server. Additional data modules can be registered by using the AstaServerSockets.OnAddDataModule event.

Since ASTA servers can use any number of data modules, in order to support the proper threading of ASTA servers typically there will be ONE TDatabase (using BDE terminology) like component per data module and additional data modules will need to be initialized with their TDataSets set to this TDatabase. Use the OnAddDataModule event to perform this initialization.

1.6.4.12 Login Process

ASTA 3 introduces a centralized login method on the server that can replace both the OnClientLogin and OnClientDBLogin and also allows for an AstaParamList to be returned to remote Clients through the new OnLoginAttemptParams event. For backwards compatibliity both OnClientLogin and OnClientDBLogin are still supported.

The following is a list of events that occur when an ASTA client connects to an ASTA Server. By default AstaClients do not go through the login process. Most production applications of course will require a login but this is not the default behavior of the AstaClientSocket. Set the AstaClientSocket.AutoLoginDlg to anything but ltNoChallenge to force a login.

1. OnClientConnect fires and the ClientSocket:TCustomWinSocket is added to the AstaServerSocket.UserList with no Username.

2. OnUserListChange fires with a tuConnect for state

3. if OnValidateSocket is assigned it is fired and the client is disconnected if the event returns a False in the IsValid:Boolean.

4. If the AstaServer.SecureServer is set to true only a Login message will be accepted next from an AstaClientSocket. If Encryption is used only messages that can be successfully encrypted will be processed by the server. If a message cannot be decrypted the client will be disconnected.

5. The AstaClientSocket starts the login process sending the UserName,Password, Application Name and Versioninformation to the server along with the ClientSocketParams. All this informatin will be available in the AstaServerSocket.UserList after a successful login.

6. The AstaServer handles the login request in either the OnClientLogin event or the OnClientDBLogin event if the server is running in PersistentSession Threading Model.

7. The Login is processed and a message is sent to the client that can be handled by the OnLoginAttempt event.
If the Login Request is denied and the DisconnectClientsOnFailedLogin:Boolean is set to true then the Client is disconnected immediately.

8. A version check is made by the ASTAServerSocket and theOnAstaClientUpdateDecide event is fired to decide if the upgrade is to be allowed. If so the OnAstaClientUpdate event is fired. If an update is requested and confirmed a new version of the Client Application is streamed down and a new version of the client application is launched.

Starting the whole login process one more time.

9. Any AstaClientDataSets opened at design time are closed and Opened after the ClientSocket connect so they can perform any selects from the remote server.

10. The AstaClientSocket then requests version information from the AstaServersocket using the Asta Remote Administrative API call of RequestUtilityInfo passing in uiServerVersion so the AstaclientSocket.ServerVersion can be update accurately.

1.6.4.13 Server Side Programming

Asta servers have been designed to support Asta client applications with no changes. Everything that you can do with a normal 2-tier fat client database application, ASTA servers can provide to ASTA client applications. There is much to be said in developing and deploying a generic ASTA server that can serve diverse client applications. Writing client side SQL select statements, calling stored procedures from ASTA clients, and using ASTA to generate SQL on update, insert, allow ASTA developers to quickly and easily deploy thin client 3 tier applications. However there may be times when you want to customize an ASTA server or to NOT use SQL in an ASTA client.

ASTA provides tools to allow you to easily create and maintain ASTA servers that will allow you to deploy ASTA client applications that require NO SQL using TAstaProviders, Server DataSets or the TAstaBusinessObjectManager.

1.6.4.14 Threading

1.6.4.14.1 Threading ASTA Servers

In order to handle a large amount of concurrent users, ASTA servers can multi-thread their database sessions. The AstaServerSocket is non-blocking and event driven so that client requests can always be handled. Performance issues are normally related to bandwidth considerations and a properly designed client application is critical if you want your ASTA server to respond promptly to requests.

In order to multi-thread any database application, each user must have their own connection or session to the database, and their own copy of the component that is to be used in conjunction with this session, usually a query or stored procedure component. Using the BDE, each client needs a TSession and their own TQueries. Using ODBC, each client would need their own handle to a database (THdbc) and their own query component (TOEDataSet using ODBCExpress).

ASTA servers can run in one of 4 different threading models: Singled Threaded, Persistent Sessions, Pooled, and Smart Threading.

It is recommended that you use ASTA SmartThreading on all ASTA 3 servers as this gives you the best of both worlds: the ability to scale the server with pooled sessions and also to have the ability to have any single user have their own specific Database Connection via a PersistentSession, and even to switch in and out of this mode.

For small numbers of users the Single Threaded model is the most efficient mode to run an ASTA Server. The most scalable is the Pooled Threading Model. If you want to be able to keep an open cursor on an ASTA server and be able to use packetized queries on ASTA clients, or use database user name and password, use the Persistent Threading Model.

ASTA servers can also maintain client database state while the client is connected to the server. This model will not scale as well, but there may be certain occasions where you would want a client to keep a persistent database connection. For example, you might want a client to use a real database name and password to keep their database security level applicable to them or when you want to open a server side cursor and fetch partial rows from the server. See Coding ASTA Servers for steps to create an ASTA server.

There are three elements of a multi-threaded database ASTA server.

1. Set the ThreadingModel property. If tmPooledSessions is chosen, then you must set the DataBaseSessions property to the number of sessions to be created at server startup.

2. Code the ThreadedDBSupplySession event.

3. Code the ThreadedDBSupplyQuery event

When an ASTA server starts up, the method CreateSessionsForDbThreads is called. If the DataBaseSessions property is greater than 0, and the ThreadedDBSupplySession event and the ThreadedDBSupplyQuery event is coded, then an AstaSessionList can be created that will act as a pool of connections to the database. If you choose to thread the server, these two events must be coded. Use the function UseThreads to verify at runtime which threading model is in effect.

1.6.4.14.2 General Threading Tips

One of ASTA's design goals was to shield the developer from any Threading issues and to allow ASTA to handle all threading internally. There are some general thread habits you need to follow which mostly apply to the VCL GUI controls not being thread safe and the use of Global Variables.

To write thread-safe code (as in the case of file I/O, make sure in each instance of the thread, you will not be writing to the same file), and also this means that the threaded procedure should not access directly the global variables (which will have no guarantee of maintaining its value in each thread) and the properties of objects in the user interface. Doing so would cause Access Violation Errors. You will find that by using global variables and setting the properties of the User Interface Visual Objects, unfamiliar access violation errors occur, and yet your code compiles perfectly. However, you can still use those global variables, and read the values of the properties of VCL objects by passing it through the TAstaParamList parameter and then decoding it within the threaded procedure.

1.6.4.14.3 Persistent Threading Model

In order to fully support JDBC clients, as well as customer requests, ASTA 2.5 will contain new features to the PersistentSessions threading model.

When running an ASTA server in PersistentSessions, each remote client will have their own database components (data module) so that the database user name and password can optionally be used. When database work is required in this threading model, each database activity is done in a thread with each client having their own database connection. (This is NOT the most scalable threading model. To scale an ASTA server, Pooled Sessions Threading is encouraged.)

ASTA servers can be started in this mode by putting PersistentSessions on the command line. When running in the Persistent Sessions threading model the following features are now supported:

1. Packet Queries:

With SQLOptions.soPackets set to True and RowsToReturn > 0, an SQL query will execute on the server but only RowsToReturn rows will be returned to the client.
With AutoFetchPackets set to True, any visual control that attempts to scroll or access EOF will force the AstaClientDataSet to call GetNextPacket which will launch a new thread on the server using the initial dataset opened on the server by the SQL statement. With AutoFetchPackets set to False you must explicitly call GetNextPacket.

2. Packet Locates

```
function GetNextPacketLocate(const KeyFields: string;
const KeyValues: Variant; Options: TLocateOptions): Integer;
```

GetNextPacketLocate works like the TDataSet.Locate but it will use the query created from step #1. Depending on your database on the server, this can be a very fast operation. Using a file server database like DBISAM, locates will use indexes if the SQL does not include a join. The GetNextPacketLocate will call a TDataSet.Locate on the server and return RowsToReturn rows, and place the server side cursor on the position where the locate has positioned it. This can result in very fast fetches on large tables. Your mileage will vary depending on how the database component used on the server handles this.

3. StartTransaction:
Normally in the n-tier model, round trips to the server should be minimized and activities and transactions should be started and ended on the server in one action. Using ApplyUpdates, ASTAClientDataSets send SQL to the server, start a transaction, execute any SQL from the client and then commit or rollback the transaction as necessary. When running Persistent Sessions you can now explicitly start a transaction with the AstaClientSocket.

```
procedure StartTransaction;
```

If the server is not running in Persistent Sessions an exception will be raised.

4. `procedure EndTransaction(Commit: Boolean);`
You can now also end transcactions from the AstaClientSocket.

5. The AstaClientDataSet has the following new methods:

```
function ParamQueryCount: Integer;
function SendParamterizedQueries: TAstaParamList;
procedure ClearParameterizedQuery;
procedure AddParameterizedQuery(TheSQL: string; P: TAstaParamList);
```

The following code shows how these can be used. The idea is that any number of parameterized queries can be created and then sent to the server in one call.

```
procedure TForm1.BitBtn1Click(Sender: TObject);
var
  p: TAstaParamList;
  i: Integer;
```

```
begin
  p := TAstaParamList.Create;
  p.FastAdd('LastName', 'Garlandx');
  p.FastAdd('EmpNo', 2);
  with AstaclientDataSet1 do begin
    ClearParameterizedQuery;
    AddParameterizedQuery('UPDATE Employee SET LastName = :LastName ' +
'WHERE EmpNo = :empNo', p);
    p.ParamByName('Empno').AsInteger := 4;
    AddParameterizedQuery('UPDATE Employee SET LastName = :LastName ' +
'WHERE EmpNo = :empNo', p);
    p.Clear;
    p.Fastadd('Phone', NewExt.Text);
    p.Fastadd('PhoneExt', OldExt.Text);
    AddParameterizedQuery('UPDATE Employee SET PhoneExt = :Phone ' +
'WHERE PhoneExt < :PhoneExt', p);
    SendParamterizedQueries;
    p.Free;
  end;
end;
```

1.6.4.14.4 ASTA Smart Threading

Prior to ASTA 3 there were 3 threading models Single, Persistent and Pooled. ASTA 3 introduces tmSmartThreading. This allows Pooled and PersistentSessions to be mixed and is the default threading model for all ASTA 3 servers.

By default remote users will share a database connection from the ASTA Database session pool. To allow for any user to get their own unique database connection, there is a new event defined on the AstaServerSocket:

```
property OnAssignPersistentSession: TAstaServerAssignPersistentSession;
```

Below is an example of the Asta3ADOServer. In this case, if a user has the name of 'PersistentSessions' then a datamodule will be created for that user.

```
procedure TServerDM.ServerSocketAssignPersistentSession(Sender: TObject;
  TheClient: TUserRecord; UserName, Password: string; var
  PersistentDataModule: TComponent; var Verified: Boolean);
begin
  if CompareText(UserName, 'PersistentSessions') = 0 then begin
    PersistentDataModule := TAstaDataModule.Create(nil);
    with TAstaDataModule(PersistentDataModule) do begin
      AdoConnection.ConnectionString :=
  FDatabasePlugin.DatabaseConnectionString;
      AdoConnection.Connected := True;
      Verified := True;
    end;
  end
end;
```

The TUserRecord, defined for each remote user, has a property of DatabaseSession: TComponent.

Below is the OnClientAuthenticate event as coded on the Asta3ADOServer. Before this event is fired, the OnAssignPersistentSession is fired. If a Session is created for that user it is assigned to the
UserRecord.DatabaseSession property. Whenever a client is access on the server you will

have access to this property.

```
procedure TServerDM.ServerSocketClientAuthenticate(Sender: TObject;
  Client: TUserRecord; UName, Password, AppName: string;
  ClientParamList: TAstaParamList; var LoginVerified: Boolean;
  var PersistentSession: TComponent);
begin
  //The ClientParamList will return to the AstaClientSocket and be
  available in the
  //OnLoginParamsEvent. You can fill it up with any values you desire.
  if Client.DatabaseSession <> nil then
    ClientParamList.FastAdd('Database',
  TAstaDataModule(Client.DatabaseSession).A
doConnection.ConnectionString)
  else
   ClientParamList.FastAdd('Database', 'Pooled' +
  FDatabasePlugin.DatabaseConnectionString);
  LoginVerified := True;
end;
```

Additionally, remote users can move from Pooled to Persistent Sessions. Below is some code from the Asta 3 SmartThreading tutorial that shows how to change a user from Pooled to persisent Sessions. There is an Extra ParamList
available that will be transferred to the server side UserRecord.ParamList. This would allow you to add logic so that the correct Database connection could be created for a specific client.

```
procedure TForm1.SessionButtonClick(Sender: TObject);
var
  Params: TAstaParamList;
begin
  Params := TAstaParamList.Create;
  //you can add what you want to the UserRecord.paramList here by adding
  the params
  try
    if InString('Pooled', SessionButton.Caption) then
      AstaclientSocket1.RequestUtilityInfo(uiSetPooledSession,
  Params.SaveToString)
    else
      AstaclientSocket1.RequestUtilityInfo(uiSetPersistentSession,
  Params.SaveToString);
  finally
    Params.Free;
  end;
end;
```

Value	Meaning
tmSingleSession	This is the default model and should be the fastest for small numbers of users or larger numbers of users that don't have excessive concurrent activity.
tmPooledSessions	This is a resource conscious scaling model. Use the DataBaseSessions property to set the number of sessions that are created at server startup. All connections are created at server startup, and client requests are paired with sessions from this pool on an as-needed basis. If you are trying to support a large number of users, this is the model to use. The pooled number increases dynamically if there is a demand.
tmPersistentSessions	This allows each client to connect to a session when the client connects to the server. The sessions are persistent. Used in conjunction with the TAstaClientSocket.Login property, you can use this threaded model to allow users to login with their actual database login, and even allow each client to access a separate alias. When the client connects, the socket triggers an event to create a database session. Any subsequent client database access will pair the incoming socket with the session chosen at startup, within a spawned thread. When the client disconnects, the session is disposed of. When an ASTA server runs in this mode, ASTA clients can fire a server query and keep a cursor open on the server. Instead of fetching all the records in the query the client can request X number of rows via the PacketSize property. See Packet Fetches for a Full discussion of Packetized Querys.

1.6.4.15 UseRegistry

The AstaBDEServer and AstaODBCServer will both respond to the UseRegistry command line switch that allows you to pull login information from the registry instead of entering it directly. On the login dialog, if you click on the Use Registry check box, the alias (or datasource), username and password will be saved in the registry. You can then subsequently run the server with the command line switch UseRegistry to start up the server and bypass the login dialog.

1. Start up an ASTA server, and then choose an alias and type in your user name and password. Make sure to click on the UseRegistry check box.
2. Run it once.
3. Next time you run the server put UseRegistry on the command line and it will use the alias, user name and password from the registry.

1.6.5 ASTA Messaging

ASTA was designed to be used by Delphi database application developers using skill sets they already possess. ASTA uses TCP/IP for it's transport and contains an easy to use messaging system. ASTA ParamList Messaging provides an extremely flexible and powerful communications framework.

ASTA uses non-blocking event driven sockets as it's underlying transport. These sockets are by their nature asyncronous. This means that a message sent by one method does not wait for any response from a server and if a response does come it will fire an event. So if you send something here, when you get a response it will be over there. ASTA was designed that you would not have to deal with foolishness such as this. Both the AstaServerSocket and AstaClientSocket have easy to use messaging calls. You can send a string or a stream with SendCodedMessage or SendCodedStream.

Here is the way the ASTAClientSocket can be used:

```
procedure SendCodedMessage(Msgid: Integer; Msg: string);
procedure SendCodedStream(MsgID: Integer; MS: TMemoryStream);

AstaClientSocket1.SendCodedMessage(1000, 'Hello World');
```

Then on the server you would just code the AstaServerSocket.OnCodedMessage like this:

```
procedure TForm1.AstaServerSocket1CodedMessage(Sender: TObject;
  ClientSocket: TCustomWinSocket; MsgID: Integer; Msg: string);
begin
  case Msgid of
   1000: AstaServerSocket1.SendCodedMessage(ClientSocket, 1001,
           'Hello back to you!');
  end;
end;
```

When we built ASTA there were times of course that we wanted to send other kinds of data than strings and streams so we created the TAstaParamList. The TAstaParamList is just like a TParamList that you are familiar with on the TQuery component but it is fully streamable. It also can contain datasets, any kind of binaries and even other ParamLists. So once we created the TAstaParamList and the call procedure SendCodedParamList(Msgid: Integer; Params: TAstaParamList) we pretty much could transport anything back and forth between servers and clients (and client to client also btw) easily.

But, you say, in the above examples we send in one place and receive in anevent and you told us that you saved us from that. AstaClientsockets have a call of SendGetCodedParamList which sends a TAstaParamList and also waits until the server sends back a response. This allows you to write procedural code rather than event driven code.

```
var
  P1, P2: TAstaParamlist;
begin
  P1 := TAstaParamList.Create;
  try
    P1.FastAdd('Hello World');
    P1.FastAdd(Now);
```

```
P1.FastAdd(4.333);
    P2 := AstaClientsocket1.SendGetcodedParamList(1010, P1);
    //Do something with the P2 ParamList that is returned
    //from the function result
  finally
    P1.Free;
    P2.Free;
  end;
 end;
```

ASTA users have thought of all kinds of ways to use ASTA messaging from taking fairly static queries and executing and compressing them on ASTA servers to be sent out using ASTA messaging, to controlling remote servers, to getting back performance information form servers, to sending faxes. Once you have a middle tier, there is a complete world that opens up of new possibilities.

See also More on ASTA Messaging

1.6.5.1 More on ASTA Messaging

Asta provides a solid and easy to use messaging system to facilitate communication between the server and client. Messages consist of an Integer and either a String, a Stream or an AstaParamList (which can contain any datatypes including DataSets). Both TAstaServerSockets and TastaClientSockets have the following message routines. The TAstaClientSocket sends to the implicit ServerSocket while the TAstaServerSocket must identify which client it is sending to.

AstaClientSocket

```
procedure SendCodedMessage(MsgID: Integer; Msg: string);
procedure SendCodedStream(MsgID: Integer; MS: TMemoryStream);
procedure SendCodedParamList(MsgId: Integer; Params: TAstaParamList);
function SendGetCodedParamList(Msgid: Integer; Params: TAstaParamList);
```

AstaServerSocket

```
procedure SendCodedMessage(ClientSocket: TCustomWinSocket; MsgID:
Integer; Msg: string);
procedure SendCodedParamList(ClientSocket: TCustomWinSocket; MsgID:
Integer; Params:TAstaParamList);
procedure SendCodedStream(ClientSocket: TCustomWinSocket; MsgID:
Integer; MS: TMemoryStream);
```

There are also three events defined on both the TAstaClientSocket and TAstaServerSocket to allow for messages to be received.

```
property OnCodedMessage: TAstaClientCodedDataEvent;
property OnCodedStream: TCodedStreamEvent;
property OnCodedParamList: TCodedParamEvent;
```

SendCodedMessage allows you to send a message to the server, and then make a custom response back to the client. The SendCodedMessage method allows you to roll your own protocol where certain MsgIDs have specific meaning to your application. The ability to create your own protocol in this fashion is a simple yet powerful technique. See the Simple Business Objects Example below.

The simple code below sends a message to the server, but the message is sent with two different MsgIDs. The server will handle the message differently depending on which of the MsgIDs, 1700 or 1750, accompanies the message.

```
procedure TForm1.Button4Click(Sender: TObject);
begin
  AstaClientSocket1.SendCodedMessage(1700, eClientCodedMessage.Text);
end;

procedure TForm1.Button3Click(Sender: TObject);
begin
  AstaClientSocket1.SendCodedMessage(1750, eClientCodedMessage.Text);
end;
```

The server responds with the following implementation code.

```
procedure TForm1.AstaServerSocket1CodedMessage(Sender: TObject; MsgID:
Integer; Msg: string);
begin
  case MsgID of
    1700: AstaServerSocket1.SendCodedMessage(MsgID, UpperCase(Msg));
    1750: AstaServerSocket1.SendCodedMessage(MsgID, LowerCase(Msg));
  end;
end;
```

When returned to the client, it is picked up in the AstaClientSocket's OnCodedMessage event. Then SendCodedStream(MsgID: Integer; MS: TMemoryStream) compliments the SendCodedMessage method. It can be used to send streams to the server and then the server can respond with a custom action. The following code demonstrates the usage.

The client sends a message to the server.

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  AstaClientSocket1.SendCodedMessage(2000, '');
end;
```

The server responds by loading a memo into a stream and sending it to the client. procedure

```
TForm1.AstaServerSocket1CodedMessage(Sender: TObject;
  MsgID: Integer; Msg: string);
var
  MS : TMemoryStream;
begin
  case MsgID of
    2000: begin
            MS := TMemoryStream.Create;
            mSelectSend.Lines.SaveToStream(MS);
            AstaServerSocket1.SendCodedStream(2000, MS);
            MS.Free;
          end;
  end;
end;
```

The client receives the stream and displays it in a memo and saves it to a file.

```
procedure TForm1.AstaClientSocket1CodedStream(Sender: TObject;
  MsgID: Integer; MS: TMemoryStream);
begin
  case MsgID of
    2000: begin
```

```
mFileFromServer.Lines.Clear;
            mFileFromServer.Lines.LoadFromStream(MS);

            mFileFromServer.Lines.SaveToFile('ATGTestFile.Txt');
          end;
  end;
end;
```

The TAstaParamList allows any number and any kind of data to be transported easily and efficiently. See the CodedParamList Tutorial for an example of using an AstaParamList to send a TextFile from a client to the server. Not only does it send the file, but the name of the file and the time it was sent. The same tutorial contains an example of sending a zip file from the client to the server, demonstrating how AstaParamLists can be used to transport binary files.

Chat Messaging
Procedure SendChatPopup(S: String) will broadcast a message from the sending client to all the other clients connected to the same server. This message is intrusive and it will stop the other clients from working while the message is displayed in a modal dialog that halts the other clients' progress. See SendChatEvent for an alternative.

The following code reads the input from a Memo named mChatMessage and broadcasts it to all the clients (including the sending client) in a popup dialog box.

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  AstaClientSocket1.SendChatPopup(mChatMessage.Text);
end;
```

Procedure SendChatEvent(S: String) broadcasts a message from the sending client to all other clients connected to the same server, but it is not intrusive.like SendChatPopup. This message must be intercepted and handled in the OnChatMessage event handler. If you do not assign the event handler, then the message will NOT be displayed.

This code broadcasts a message from the client to all the other clients.

Simple Business Objects Example
The following code shows how you could send CodedMessages to a business object instantiated on the server. In this scenario, 1800 and 1850 are "protocol codes" that you control. The client can call those codes, with or without a parameter and invoke a custom response from the server.

```
procedure TForm1.AstaServerSocket1CodedMessage(Sender: TObject;
  MsgID: Integer; Msg: string);
begin
  case MsgID of
    1800: begin
            if MyBusinessObject.ChargeSalesTax(StrToInt(Msg)) then
              AstaServerSocket1.SendCodedMessage(MsgID,'TRUE')
            else
              AstaServerSocket1.SendCodedMessage(MsgID,'FALSE');
          end;
    1850: begin
            AstaServerSocket1.SendCodedMessage(MsgID,
```

```
MyBusinessObject.NewCreditLimit(Msg));
          end;
  end;
end;
```

1.6.5.2 ParamList Messaging

ASTA has a flexible and easy to use Messaging Layer. AstaParamLists can contain any kind of Data including other ParamLists and DataSets. There are tutorials available to show how to pack up ParamLists containing other paramlists and DataSets.

ASTA of course has a DataBase layer but there are a surprising number of ASTA applications deployed that only use ASTA messaging. As a Database Application Developer, you world will change forever once you learn the power of ASTA ParamList messaging. Messages can be send between client and server and also directly to other clients either through the server or by creating an TAstaServerSocket on remote clients and allowing for Peer to Peer Access (P2P).

ASTA supports not only users connected to a server or connected to each other via P2P but disconected messaging with the Asta MessageQueue which allows for messages to be send to users are are not connected to a server. Messages are stored on the server and when users connect the messages are picked up.

ASTA ParamLists are supported cross with libraries written in C++ and java for support of Palm, Wince, GCC Linux, Java and AstaCOmClient.dll for Win32 and WinCE. See the SkyWire help for more information on non-vcl implementations of AstaParamLists and the TDataSet like data structure called a DataList.

ASTA 3 ParamList messaging is implemented on the AstaClientSocket and AstaServerSocket. Internally ASTA uses ParamLists extensively to send disparate data between client and server. ASTA ParamList messaging is the basis for other features such as:

1. ASTA Instant Messaging API (IMA)
2. ASTA Asta FTP (AFTP) ASTA file transfer protocol that allows for large files to be transfered in configurable "chunks"
3. ASTA File Synchronization- Cross Platform File Synchronization (Win32 and WinCE)

AstaClientSocket
The AstaClientSocket has calls to SendCodedParamLists to server and to optionally wait for a response. You must decide if you require your client to block or wait for any response from a server or, to code using async calls that allows for server side processes to be "initited" by a messaging call and when completed to return to the client. Note: Asta Server Methods support async calls with the DelayAction:Boolean option on the TAstaBusinessObjectsManager.

ASTA sockets are async event driven sockets (jump to async link) and by nature do not block. We have added blocking calls to allow you to write procedural code. So it's up to you to decide which calls to use.

Non-Blocking Calls
SendCodedParamList is a non-blocking call and the building block for all ASTA

messaging. You create a TAstaParamList, fill it with data and send it to the server with a Msgid:Integer. On the server, the ParamList will be received in the OnCodedParamList call of the AstaServeSocket.SendGetCodedParamList will not block or wait so after the server processes the call:
SendCodedParamList - non blocking
SendNamedUserCodedParamList - requires used to login with unique usernames. Then you can send a named message to 'shaq.lakers' or 'jason.nets' on the server. This is used extensively in the Asta Instant Messaging API.

Blocking Calls
These calls block or "wait":
SendGetCodedParamList
SendGetCodedDBParamList
GetCodedParamList
event OnCodedParamList
This event is fired when a the server calls SendCodedParamList or ClientBroadcastParams on the server.

AstaServerSocket
Methods
SendCodedParamList - This is the base call to send ParamLists to ASTA clients. If the client is a skywire client, (palm,wince,java) any translations are done internally so the client can receive the paramlist.
ClientBroadcastParams - this allows for paramlists to be send to all users connected to the server.

Events
Asta uses non-blocking async sockets so that ASTA servers don't require threads for messaging calls. ASTA implemented the autoupdate feature using ParamList messaging so that when the SendGetCodedParamList call is made from an ASTA client, the return of any data is threaded. This applies only to the return of data.

The recommend way to code ASTA 3 servers for SendGetCodedParamList support is to code the OnOutCodedParamList.

If you want to support db calls in ASTA messaging use OnOutCodedDBparamList. If you have a long running process that you want to thread, then use the ThreadedUtilityEvent which launches a thread for your process. This should ONLY be used in async alls
(sendCodedParamList) from the client.

OnCodedParamList - This is what is fired when a client calls SendCodedParamList. in ASTA 2 SendGetCodedParamList also flowed through this event but with ASTA 3 there is a new event OnOutCodedParamList that allows you to code your servers so that OnCodedParamList handles Async calls and OnOutCodedParamList handles all blocking calls.

OnOutCodedParamList - This is the recommended way to handle SendGetCodedParamList or GetCodedParamList calls.

OnOutCodedDBParamList - This is what needs to be coded on the server and is the server side of sendGetCodedDBparamList. This event will be run in a thread created for

you by ASTA and will pass in a datamodule so you will have access to any database component from your ASTA datamodule.

PDASendParamList - sends a ParamList to a SkyWire Client. SendCodedParamList will map to this call for a SkyWireClient

PdaServerPlugin
See the SkyWire documentation on how to handle SkyWire (non-vcl) ParamList messaging on servers.

Threading Messaging Calls
If you want to thread messaging calls on your server, you can either use ThreadedUtilityEvent or ThreadedDButilityEvent. The ThreadedDButility event will allow you to be all async, using the SendCodedParamList call from the client. If the client wants to block use SendGetCodedDBParamList

The table below recaps all the ASTA ParamList messaging calls.

	Client Call	**Server Event**	**Server Method**
Async no Database	SendCodedParamList	OnCodedParamList	SendCodedParamList
Async no Database	SendNamedCodedParmList	OnCodedParamList	SendCodedParamList
Blocking no Database	SendGetCodedParamList	OnOutCodedParamList	None required
Blocking No Database	GetCodedParamList	OnOutCodedParamList	None required
Blocking with Database	SendGetCodedParamList	OnCodedParamList	None required
Async Threaded	ThreadedUtilEvent	OnCodedParamList	ThreadedUtilityEvent and call SendCodedParamList in the event you pass in
Async DB Threaded	ThreadedDB	OnCodedParamList	ThreadedDBUtilityEvent and call SendCodedParamList in the event you pass in
Server Broadcast	OnCodedParamList		ClientBroadCastParams

1.6.5.3 Transfering Large or Groups of Files

ASTA supports sending and receiving any kind of data including files using ASTA messaging. There are examples and tutorials that use SendCodedStream or SendCodedParamList. This is all easy and straightforward but there are times when you want to send very large files or lots of files. If that is the case, then you can use some of the ASTA FileSegment Send routines that allow you to configure the size or block sizes of data sent over the wire.

AstaParamlist messaging is very flexible and should be used whenever possible. When calling SendGetCodedParamlist from AstaClients, any sends will be sent in a background thread. But sometimes, if you are sending very large files (10mb) you may want control on how much processing time you want the server to commit to sending and receiving files.

In order to support this, ASTA has routines to send files from the server to the client or from the client to the server in configurable "chunks".

There are 3 ASTA Tutorials that cover sending files

1. FileSend- This shows how to use normal ASTA messaging to send files between client and server.

2. File Transfer : ServerToClient - This shows how to request files from a remote server and send them in a thread in configurable "chunks"

3. File transfer : Client To Server - this shows how to send a file or groups of files using masks to a remote server.

1.6.6 ASTA Tools

1.6.6.1 Asta Anchor Server

ASTA clients typically connect to an ASTA server using an IP address and port number. ASTA servers scale quite well but there may be times when you want to have fail over abilities with multiple servers, or just have too many clients to be handled by one server. The ASTA Anchor Server provides load balancing and fail over abilities. Instead of clients connecting directly to a remote ASTA server they connect to the AnchorServer which then finds the next available ASTA server and sends back to the client the IP address of that server. The client disconnects from the Anchor Server and connects to the ASTA server designated by the Anchor Server.
The Anchor Server can be configured to connect to any number of ASTA servers. When the Anchor Server starts up it will attempt to connect to all the ASTA servers and will display the current status of each server. When the Anchor Server starts, it can show the connected number of users for each ASTA server.

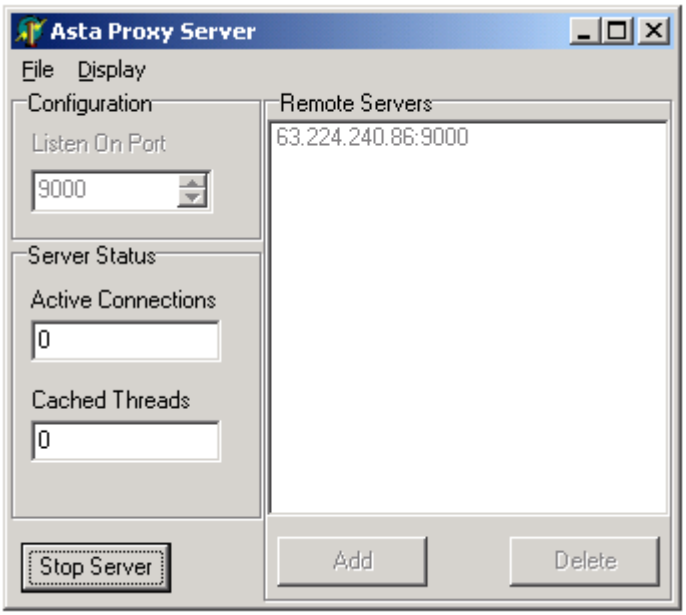
If an ASTA Server goes down, all the remote users from that server will be disconnected and will then connect back to the Anchor Server which will hand them off to other ASTA servers. This is the concept meant by fail over. One of the advantages of the N Tier architecture is that once you design your ASTA client application it can scale simply by adding more ASTA servers to be used with the ASTA Anchor Server.

The AstaAnchor server makes use of the Asta Adminstrative Remote API to retrieve information from remote ASTA Servers. Since the ASTA Anchor Server requires multiple ASTA servers it is not part of the ASTA Entry suite but is part of the ASTA Power Pack and higher licenses.

1.6.6.2 ASTA Conversion Wizard

1.6.6.3 ASTA Proxy Server

ASTA proxy server can be requested by any ASTA server and can be run on a client firewall to allow Client applications to have their ports redirected by the Proxy.



1.6.6.4 AstaServerLauncher

The AstaServerLauncher allows multiple ASTA servers to be managed from one executable and if they happen to go down, the AstaServerLauncher will "bounce" them back. Future versions of the AstaServerLauncher will have a remote interface to allow for remote administration.

The AstaServerLauncher can be run as an NTService using the AstaServerLauncherNTS This allows any ASTA Server to be controlled by an NT Service.

1.6.6.5 AstaSeverLauncherNTS

The AstaServerLauncherNTS is an NT Service version of the AstaServerLauncher.

This does not have a user interface, it is run as a service and reads the registry to tell which programs to start. Use the AstaServerLauncher to setup the Registry configuration.

To install it as a service, run the program with the command line parameter of /Install (you can create a shortcut to use for this by editing the shortcut's Target property and adding /Install to it, /Uninstall does the reverse). Then in control panel, open up Services. Look for AstaServerLaunherNTS. Double click it. Set it to Manual or Automatic start. Click on Allow Service to Interact with Desktop so that it is checked. (You can also set it to run in its own session by selecting a particular account with This Account radio button). Press OK.

You can start the service manually. Once started it will read the registry and start up any programs that need AutoStarted. It will then keep these programs alive if they are set to AutoReStart.

1.6.6.6 AstaSQLExplorer

The ASTASQLExplorer is an ASTA client application available with full source that can show ASTA server side information including metadata, ASTA Providers and ASTA ServerMethods.

AstaSQLExplorer has five main tabs:

1. SQL
Enter a free format sql query. Certain TDataset descendants allow you to do non-select queries. Execute the query. Once you have executed the query, you must specify/set the primary keys and an update table. To do this, click on the "Set Primary Keys" button or select the menu item. A dialog box will be displayed that will list the tables of the query in a combobox. Select the Update Table. If the table that you select contains any primary keys, as returned by the Asta server (not all Asta servers return primary key information), the primary key fields will be checked. You can change the fields to be used as the primary keys for edits, by checking the fields. Select OK. If you have selected any primary keys for the Update table, The dbnavigator will now display more buttons. You can now do edits on the result set.

2. Data
Data is a quick way to view data in a table. Select the table in the explorer tree and then select the Data tab. Unlike the SQL tab where you can specify the primary keys, ALL the fields will be used as the primary keys.

3. Params
Use this tab to display input/output parameters for a selected Business object or Provider. Select the Business object or Provider and then click on the "Retrieve parameters" button. If there are any parameters, they will be displayed in the scrollbox. Fill in the parameter values and Execute.

4. SQL Errors
All SQl errors returned from the server will be displayed on this tab.

5. History
After every query (SQL tab), the sql statement will be appended to the history dataset. Select the record to display the sql statement in th ememo field, or double click to open the query in the SQL tab. To clear existing history, delete the file AstaSQLExplorer.dat in the directory of AstaSQLExplorer.Exe

The Explorer treeview has a popup menu that will be enabled when you select a System Table, Table or View. Extract SELECT statement - Extracts a SELECT field1, field2... FROM table for the selected table. Extract SELECT and Run - Extracts a SELECT field1, field2... FROM table for the selected table AND execute the query. Not all Asta servers return mdVCLField information, so these two menu options might not return any sql statement. In this case you can change the code return just a SELECT * FROM table statement.

Note that not all servers return all the MetaData information, so not all the treeview nodes might return any data.

Some of the Astaservers return different field names for metadata. For this reason an option is added in which you can "map" the field names to be used by the explorer components. Tools|Options|Field Names. These will default to the Asta BDE server's field

names.

1.6.7 Automatic Client Updates

ASTA provides you with the ability to automatically update AstaClients. Clients must be enabled to Login to the server and for the AutoClient Update process to be activated. When you deploy your original ASTA Clients, fill in the AstaClientSocket.ApplicationName and ApplicationVersion properties. When you wish to deploy a new client, increment the ApplicationVersion (if your first distribution is 1.0, then you next release might be 1.1 or 2.0). After you have produced the new client executable (with the incremented ApplicationVersion), you can register that executable at the AstaServer. When a user of version 1.0 logs into the server, he or she will be updated automatically to the latest version of your software.

Also see AstaClientSocket's ApplicationName and ApplicationVersion properties.

There are two events on the AstaServerSocket that fire when a client comes in for an update check:
OnAstaClientUpdate
OnAstaClientUpdateDecide

Update information can be maintained remotely using the
Asta Remote Adminstrative API.

1.6.7.1 Asta Remote Administrative API

Information about Client exes that have been registered on AstaServers for version updates can now be obtained from remote clients using the new RequestUtilityInfo call from the AstaClientSocket. This API was first developed to support the ASTA Anchor Server which does load balancing and fail over.

This is an asyncronous request that will bring back a TAstaDataSet that contains information about current Client Exe's that have been registered on an ASTA Server for the autoclient update feature.

You first call AstaClientSocket1.RequestUtilityInfo(uiAutoClientUpgradeInfo, '') and a TAstaDataSet will be returned as part of the Params from the AstaClientSocket.OnServerUtilityInfoEvent.

These Datasets are stored in the TAstaParamList as Strings and they can be converted to Datasets using StringToDataSet (from astadrv2.pas) but you are then responsible for freeing the Dastaset.

```
procedure TForm1.BitBtn1Click(Sender: TObject);
begin
  AstaClientSocket1.RequestUtilityInfo(uiAutoClientUpgradeInfo, '');
end;


procedure TForm1.AstaClientSocket1ServerUtilityInfoEvent(Sender:
  TObject; MsgID: integer; Params: TAstaParamList);
begin
```

```
  case MsgID of
    Ord(uiAutoClientUpgradeInfo),ord(uiPublishedDirectories): begin
        DataSource1.DataSet.Free;
        DataSource1.DataSet := StringToDataSet(Params[0].AsString);
      end;
  end;
end;


procedure TForm1.FormDestroy(Sender: TObject);
begin
  DataSource1.DataSet.Free;
end;


procedure TForm1.BitBtn2Click(Sender: TObject);
var
  P: TAstaParamList;
begin
  if DataSource1.DataSet=nil then raise EDataBaseError.Create('You need
  fetch a dataset first!');
  if DataSource1.DataSet.state in [dsedit, dsinsert] then
  DataSource1.DataSet.Post;
  P := TAstaParamList.Create;
  P.FastAdd(DataSetToString(TAstaDataSet(DataSource1.DataSet)));
  AstaClientSocket1.RequestUtilityInfo(uiWriteAutoClientUpgradeDataSet,
  P.AsTokenizedString(False));
  P.Free;
end;


procedure TForm1.BitBtn3Click(Sender: TObject);
var
  M: TMemoryStream;
  P: TAstaParamList;
begin
  if not OpenDialog1.Execute then exit;
  M := TMemoryStream.Create;
  M.LoadFromFile(OpenDialog1.FileName);
  P := TAstaParamList.Create;
  P.FastAdd('c:\Test.exe'); //you have to know where you want to shove
  it to the server
  P.FastAdd('');
  P[1].AsStream := M;
  M.Free;
  AstaClientSocket1.RequestUtilityInfo(uiSaveClientUpgradeExe,
  P.AsTokenizedString(False));
  P.Free;
end;


procedure TForm1.BitBtn4Click(Sender: TObject);
begin
  AstaClientSocket1.RequestUtilityInfo(uiPublishedDirectories, '');

end;

procedure TForm1.BitBtn3Click(Sender: TObject);
var
```

```
  M: TMemoryStream;
  P: TAstaParamList;
begin
  if not OpenDialog1.Execute then exit;
  M := TMemoryStream.Create;
  M.LoadFromFile(OpenDialog1.FileName);
  P := TAstaParamList.Create;
  P.FastAdd('c:\Test.exe'); //you have to know where you want to shove
  it to the server
  P.FastAdd('');
  P[1].AsStream := M;
  M.Free;
  AstaClientSocket1.RequestUtilityInfo(uiSaveClientUpgradeExe,
  P.AsTokenizedString(False));
  P.Free;
end;
```

1.6.8 Firewalls

1.6.8.1 Using ASTA through a Firewall

Firewalls are a fact of life at many companies. Firewalls block access in a number of ways: by blocking ports, by filtering what kind of data can go through some port numbers, by blocking all data except through a "proxy", etc. ASTA 2.6 has a number of techniques that you can use in your ASTA client application to work through a firewall, including the addition of WinINet support. Another feature of ASTA 2.6 is that ASTA servers can now serve TCP/IP and stateless http remote clients with no changes on the server.

ASTA Servers require a static IP Address and a "port" to run on. There are 65535 ports available on any machine with port numbers lower than 1024 generally reserved for the operating system. HTTP Servers typically run on port 80, ftp runs on port 23 and SMTP (mail) servers run on port 21. You can run as many as ASTA servers as your hardware allows, on any one machine, as long as each one runs on a different port. Figure 1 shows a typical ASTA server connected to a database, usually over an Ethernet network, with remote clients connected to it via TCP/IP. The only requirement for the remote clients is that they can connect to the ASTA server IP Address and port.
The IP Address can be specified either numerically (1.2.3.4) or as a hostname (somecomputer.companyname.com).

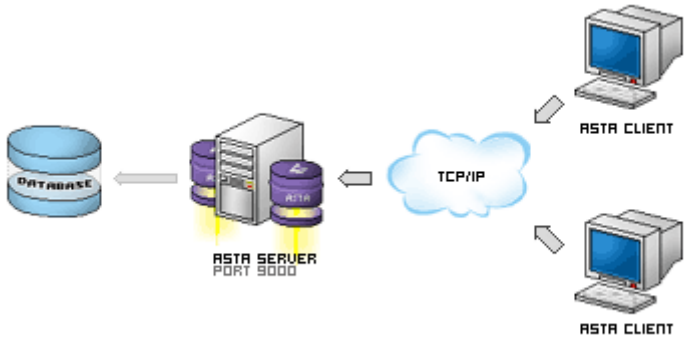


Figure 1

ASTA provides numerous ways to defeat firewalls. They can be grouped between those that maintain state and use TCP/IP and those that are stateless and use http.

State: TCP/IP and HTTP

ASTA Servers and traditional clients use TCP/IP, which maintains state. This means that once a client connects to a server, that connection is maintained until the client disconnects, the server disconnects the client or there is a network failure. HTTP on the other hand is said to be Stateless. Browsers make html requests to remote HTTP Servers and the request is completed and the browser is disconnected. Cookies are a way for HTTP Servers to maintain some kind of active user list from remote clients that are not connected to the server. There is usually some kind of time out value so when a request is made to a server a unique valued is added to a server side user list and a "cookie" containing this value is planted on the remote machine. All subsequent requests from that specific remote user will contain the value from the "cookie" so that they can be matched up on the server. Since HTTP does not maintain state, server side HTTP applications like shopping carts needs to maintain that state themselves in order to function.

This means that concepts like the AstaServerSocket.UserList and Server "push" with Server initiated messages are not possible when running stateless. After ASTA 2.6 was released, an AstaStatelessUserList was implemented that allows stateless http clients to have access to a server side UserList that implements Cookies and the concept of expiration.

Techniques that maintain State

- Have the Firewall Administrator open the port your ASTA server is running on
- Run your ASTA server on port 80 or port 8080
- Use ASTA SOCKS Support
- ASTAProxyServer

Stateless Techniques

- Set the AstaClientSocket.WinINet property to true along with ISAPI DLL on a Web Server
- Use the AstaClientSocket ability to format messages as http through an ISAPI DLL on Web Server
- Use ASTA Proxy Server support

1.6.8.1.1 Maintain State

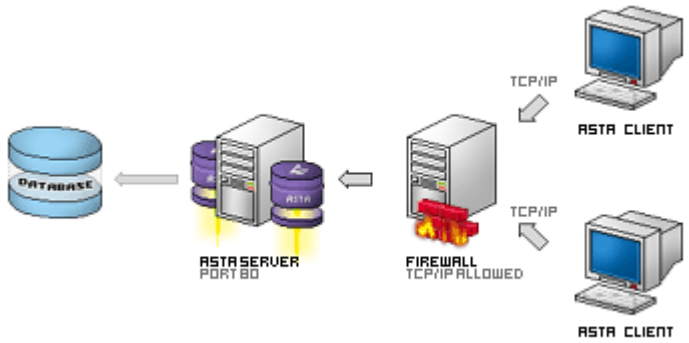
1.6.8.1.1.1 Solution #1: Open a Port

Most firewall issues are port restrictions. ASTA clients can connect seamlessly through a firewall if the administrator agrees to open up the port that the ASTA server is running on. For example, if the ASTA server is configured to use port 9000 and administrator opens up the port on the client firewall to allow TCP/IP traffic on port 9000, the firewall issue is resolved.

1.6.8.1.1.2 Solution #2: Run the ASTA Server on Port 80

Sometimes the firewall administrator will not open a port up. The next solution is to run your ASTA server on port 80 or port 8080, as firewalls must have port 80 open if clients are allowed to use browsers to access the Internet and remote HTTP servers. They must

allow unfiltered TCP/IP traffic on port 80/etc. for this solution to work. Figure 2 shows a network where there is a firewall but only port 80 is opened.



1.6.8.1.1.3 Solution #3: SOCKS Support

Running ASTA clients stateless doesn't allow you to use such features as Server broadcasts or client-to-client messaging along with provider broadcasts. SOCKS is a technology available to provide Authenticated Firewall Traversal. If you have a SOCKS4 or SOCKS5 Server, ASTA can allow you to connect via the SOCKS server and use TCP/IP as a normal client application. The AstaClientSocket has a method to set it up to connect through a SOCKS5 Server.

```
procedure SetupForSocks5Server(AstaServerAddress,
  TheSocks5ServerAddress, TheSocksUserName, TheSocksPassword: string;
  AstaServerPort, TheSocksServerPort: Word);
```

Figure 3 shows the SOCKS Setup form that allows client applications to configure their SOCKS server settings.

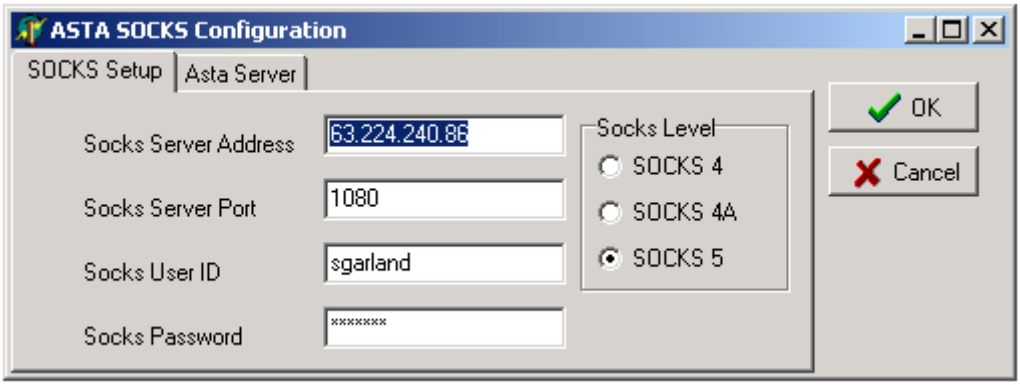


Figure 3

1.6.8.1.1.4 Solution #4: ASTA Proxy Server

ASTA also provides an ASTA Proxy Server that can be run on the same machine as the firewall that allows ASTA clients to connect to the AstaProxyServer and be re-routed to a remote ASTA server.

Figure 4 shows the ASTA proxy server.

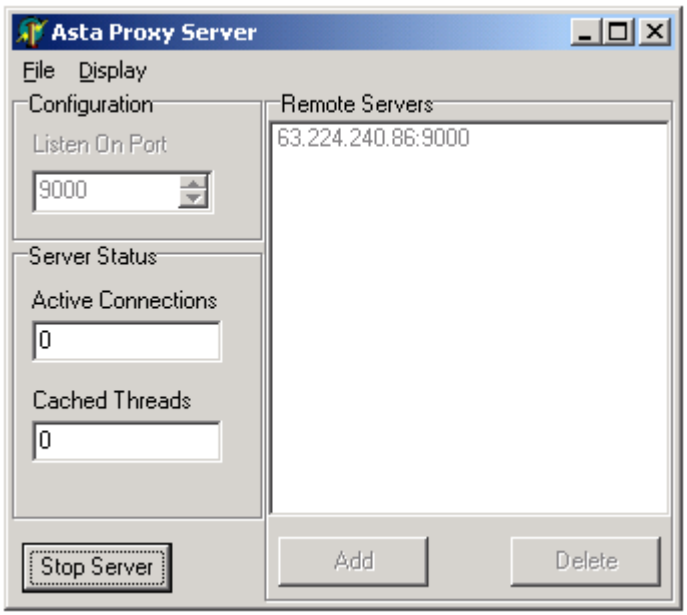


Figure 4

1.6.8.1.2 Stateless Firewall

1.6.8.1.2.1 Solution #1: Use WinInet (Highly Recommended)

The Microsoft WinInet DLL comes with Internet Explorer and provides client access to HTTP support including SSL, SOCKS and access through Proxy Servers. ASTA 2.6 allows the AstaClientSocket to use WinINet and the AstaHttp.dll to get through any Firewall that Internet Explorer can use the same registry settings for proxy servers and authentication as set by Internet Explorer. To activate WinINet support just set the public AstaClientSocket.WinINet:Boolean property to true. You would still need to setup the address of the remote Web Server and location of AstaHttp.dll as explained in Solution #2: HTTP Stateless with IIS running remotely but you would not need to set any proxy server addresses or Proxy Username or Passwords as WinINet handles this transparently.

1.6.8.1.2.2 Solution #2: HTTP Stateless with IIS running remotely

In this scenario a Web Server like IIS (Internet Information Server) receives requests from remote ASTA clients through an ISAPI dll (AstaHttp.dll) and proxies the request to an ASTA server that can be located anywhere. ASTA supplies an ISAPI dll that can be placed in the scripts directory or equivalent of the Web Server and remote clients are configured to format their messages as HTTP messages by calling SetupforIsapiUse.

```
procedure TAstaClientSocket.SetForIsapiUse(WebServerAddress,
  AstaServerAddress, AstaIsapiDll: string; WebServerPort,
  AstaServerPort: Word);
```

Clients can call AstaIsapiSetup to input the address and port information for the remote Web Server and ASTA server as well as the location of AstaHttp.dll Figure 4 shows the setup form that appears when the AstaClientSocket. AstaIsapiSetup routine is called. A Kylix built Apache DSO that does the same on Linux will be available in September 2001.

ASTA Palm and WinCE clients will also be able to take advantage of this technique. For more information see www.astawireless.com.

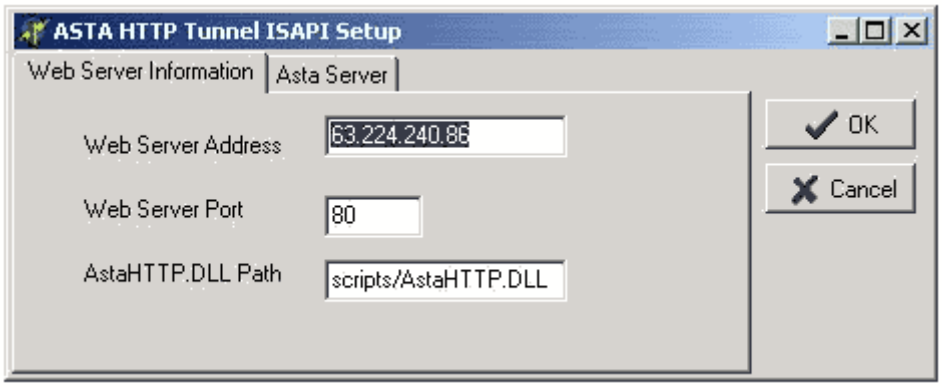


Figure 5

Figure 5 shows how ASTA clients can be configured to appear as normal browsers with HTTP formatted messages and running through an existing HTTP Server like IIS using the ASTAHTTP.DLL

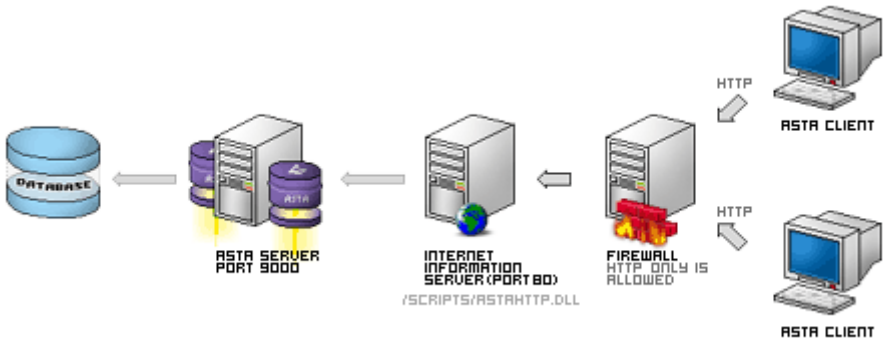


Figure 6

This is of course another stateless solution since the client is using real HTTP and communicating through IIS just like a normal browser. Use this technique if your clients don't have WinInet.dll available.

1.6.8.1.2.3 Solution #3: Through a Proxy Server

Sometimes there may be a proxy server like Netscape Proxy Server running on your client application. In this case your ASTA client application must connect to the proxy server rather than the ASTA server. ASTA supports this with the AstaClientSocket Method SetForProxyUse.

Note: WinINet is still recommended before this technique. Use this only if WinINet.DLL is not available on your client machines.

```
procedure SetForProxyUse(AstaServerAddress, ProxyIPAddress: string;
  AstaServerPort, ProxyPort: Word);
```

Figure 7 shows an ASTA proxy setup form that allows your ASTA client application to be configured to connect through a Proxy Server like Netscape Proxy Server.

Note: this call is not required if you use the ASTA WinINet support as WinINet will use the proxy settings as set by Internet Explorer.

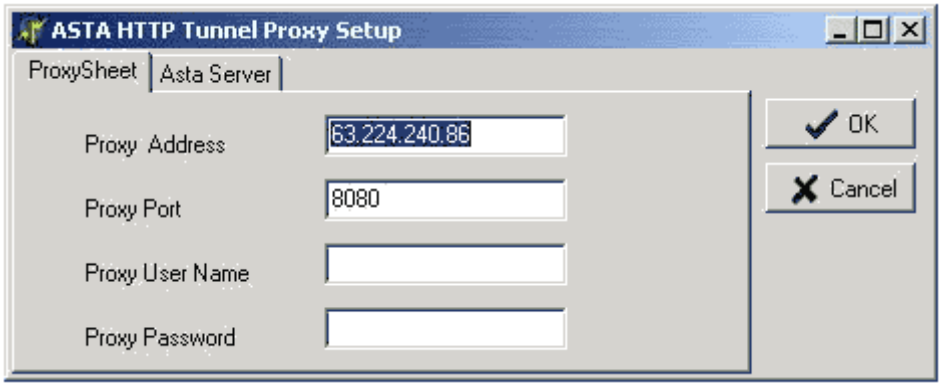


Figure 7

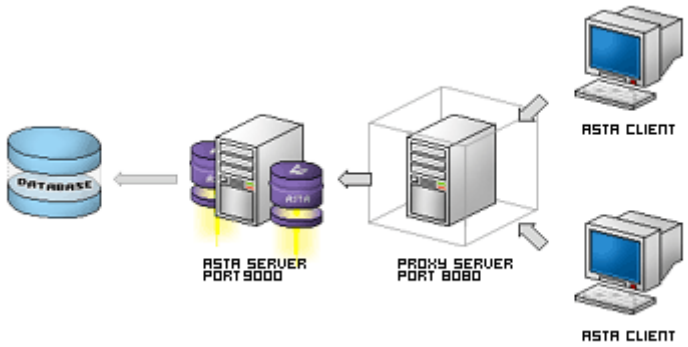


Figure 8

1.6.8.2 Stateless Clients

ASTA supports HTTP clients using the AstaClientSocket. The AstaClientSocket.WebServer property is a property that allows the configuration and control of HTTP settings. The AstaClientSocket can optionally use the Microsoft WinInet.dll that comes with Internet Explorer. WinInet uses any registry settings defined for Internet Explorer which include any proxy and Socks settings. Setting the WebServer.WinInet property (Boolean) to true is the recommended way to traverse firewalls with ASTA. If Internet explorer can browser the Internet, your ASTA WinInet enabled client will be able to communicate with your remote ASTA server through a web server and Astahttp.dll

Normal TCP/IP ASTA clients can support "server push", authentication, and ASTA messaging. When running stateless HTTP the landscape changes a bit. ASTA

does provide features to allow stateless HTTP clients to use features that ASTA users have been accustomed to.

Authentication: Stateless Cookies

Normal ASTA TCP/IP client applications support a login process that allows for remote users to be authenticated against ASTA servers with a username and password, and to also send extra params to the server and receive params back from the server on the OnLoginParamsEvent. In addition there is a UserList that maintains Username, Application Name, Connect Time and other information on the AstaServerSocket. When running stateles of course there is no UserList. ASTA supports stateless UserLists with a binarytree available as the AstaServerSocket.StatelessUserList or with the AstaMessageQueue component.

The ASTA Cookies tutorial has an example server and client to show how authentication can be done with ASTA and stateless clients.

Asta stateless support via HTTP requires a web server to run like Microsoft IIS (internet Information Server). Another good choice for testing is the Omini Web Server available from www.omnicron.ca.

1. Start the WebServer.
2. Copy AstaHttp.dll into a directory that has the rights to execute an ISAPI dll. For IIS the default is the scripts directory.
3. Compile and Run the AstaHttpTestServer that comes with the ASTA Cookies tutorial.
4. Compile and run the AstaHttpTestClient.dpr that comes in in the Asta Cookies Tutorial.

Server Side

The AstaServerSocket has an AddCookie property which is of type TAstaServerCookieOption.

TAstaServerCookieOption = (coNoCookie, coStatelessUserList, coUserCoded);

In order to tunnel via HTTP, ASTA binary messages are wrapped using the HTTP format so that they can be disguised as normal browser traffic in order to traverse any client firewall. When cookies are used, additional authentication information is included in the ASTA messaging format.

Value	Meaning
coNoCookie	No cookie information
coStatelessUserList	A binary tree is used to authenticate clients and maintain cookie information. The AstaServerSocket has a property of StatelessUserList of type TAstaStatelessUserList.
coUserCoded	The user is responsible for implementing their own cookie mechanism. in the Cookies tutorial the AstaMessageQueue Manager is used for this.

In the tutorial, the AddCookie property is set to coStatelessUserlist. Since HTTP is a stateless protocol, there is no real user list for normal ASTA TCP/IP clients. When users connect, they will provide username and password information so that the server can authenticate them (this happens in the AstaServerSocket.OnClientAuthenticate event). A "Cookie" or Cardinal value will be returned to the client that can be used in future accesss of the server. This allows for a fast/efficient way to authenticate remote clients on each connect to the server.

1.6.8.3 ASTA HTTP Tunneling

ASTA clients can have their messages formatted as HTTP if the client firewall will only allow HTTP traffic on port 80 and the remote ASTA server is running on the same machine as IIS or another web server. ASTA supports HTTP tunneling running stateless through an isapi dll sitting on the server that will pass all ASTA client/server messages through the web server to the ASTA server, and back to the client

Stateless tunneling is performed through an HTTP server using the isapi dll - ASTAHTTP.DLL. ASTA supplies an isapi DLL called AstaHttp.dll which you put intp your IIS scripts directory or
whatever directory your web server expects to find isapi dll's.

To set the server, up you must set the ASTAProtocol to AstaIsapi or use the isapi command line switch: AstaBDEServer.exe isapi

On the client you must call the AstaClientSocket.SetForIsapiUse.

```
procedure SetForIsapiUse(WebServerAddress, AstaServerAddress,
AstaIsapiDll: string; WebServerPort, AstaServerPort: Word);
```

Example

```
SetForIsapiUse('63.224.240.82', '63.224.240.84', 'isapi/AstaHttp.dll',
80, 9000);
```

The ASTA client application will then create actual HTTP messages and connect to your remote web server which will pass any messages through to your ASTA server via the ASTAHTTP.DLL and stream results back as normal HTTP messages to the client application.

To allow the user to configure the settings you can call:

```
procedure AstaIsapiSetup(ClientSocket: TAstaClientSocket);
```

See Also:
Firewalls

1.6.8.4 Setting Up IIS

I had a lot of problems with HTTP Tunneling, but that came from my IIS setup. So I uninstalled all patches and IIS, and then reinstalled IIS without any patches.

After enjoying HTTP tunneling (which is very good and fast) without any patches, I began to install all the patches one by one, and to test my app after each install.

The problem came from "URLScan", which blocks all requests containing "." considering that in /scripts/AstaHttp.dll/Asta, the dot is not an extension event if ".dll" is allowed. So users using IIS and URLScan have to modify the default configuration of URLScan, by allowing the dot, and the ".dll" in the request.

The modification to do in the URLScan.ini (look in C:\WINNT\system32\inetsrv\urlscan\):

```
[options]
UseAllowExtensions=0        ; if 1, use [AllowExtensions] section, else use
    [DenyExtensions] section
AllowDotInPath=1            ; if 1, allow dots that are not file
extension
```

Verify that .dll is not in [DenyExtensions]

Don't try to set UseAllowExtensions to 1, to deny all extensions except those allowed (I think that it is better to secure a server, but Microsoft disagrees), because in this case you'll deny requests without extensions! www.beaconseil.com will be rejected, because there's an extension in this
request. www.beaconseil.com/index.html will be allowed!

So don't modify anything, except UseAllowExtensions and AllowDotInPath!

Salutations,

Olivier MERLE
Bea Conseil

1.6.9 Provider Broadcasts

When coding applications like auctions, sometimes it is necessary to have ASTA servers "push" out information with it being directly requested from ASTA clients. TAstaProviders provide a way to do just this with little or no code.

TAstaProviders can be used to allow remote ASTA clients to be notified of any changes to any table. When using a TAstaClientDataSet, instead of using SQL, choose a Provider by pulling down the ProviderName property. Then click on the ProviderBroadcast property and set the RegisterForBroadcast property to True. If you want the AstaClientDataSet to be editable set the SQLGenerateLocation to gsServer.

When the TAstaClientDataSet is opened, it will fetch any results from the TDataSet attached to the provider on the server and also internally call RegisterProviderForUpdates which registers that TDataSet to receive notification when any other TAstaClientDataSet makes any changes to that TDataSet attached to the TAstaProvider on the server.

Any changes will be broadcast from the server side TAstaProvider and recieved in the TAstaClientDataSet. OnProviderBroadCast event. Providers can limit broadcast to specific row criteria by coding the AstaProvider. OnBroadCastDecideEvent. Providers can be networked to allow them to broadcast to all connected clients using the same TAstaProvider.UpdateTableName by calling TAstaServerSocket.NetWorkProviderBroadcast

Manual broadcasts are supported also.

1.6.9.1 Manual Provider Broadcasts

ASTA Provider broadcasts provide an almost no code, efficient way to keep make sure that remote clients receive any changes posted to the server in real time by other users. Broadcasts are triggered by AstaClientDataSets applying their updates on the server either by calling ApplyUpdates or SendProviderTransactions. There may be times however when you want to flow custom information or information triggered by client side messaging or a server side process through the provider broadcast mechanism.

ASTA now allows that to occur by using the AstaServersocket.BroadCastProviderChanges method.

```
procedure TAstaServerSocket.BroadCastProviderchanges(ClientSocket:
  TcustomWinSocket; TheProvider: TAstaProvider; const ProviderName,
  UpdateTableName: string; D: TAstaDataSet; BroadCastsToOriginalClient:
  Boolean);
```

Value	Meaning
ClientSocket	If you want to use ASTA messaging to trigger a broadcast manually you can pass in the ClientSocket:TCustomWinSocket that is available on ASTA server whenever a client makes any server request. If you want to trigger the broadcast from the server just pass in NIL for the ClientSocket.
TheProvider	You don't have access to the actual provider so pass NIL for TheProvider.
ProviderName	You must pass in the ProviderName.
UpdateTableName	If it is a network broadcast then the UpdateTableName is required.
D	This can contain any dataset you want but most probably will be a TAstaDataSet (in memory DataSet) that you create and populate. Remember, that remote clients are going to try and do updates based on the PrimeKey field and are expecting certain fields. Of course if you are coding the OnProviderBroadcast of the AstaClientDataSet yourself, do as you please and send over anything you want! The dataset must contain the same extra 2 fields as the normal broadcast dataset 'delta' and 'bookmark' where delta contains one of TDeltaType = (dtEdit, dtDelete, dtAppend, dtAppendAndDelete);
BroadCastsToOriginalClient	This only applies if you are passing in the ClientSocket parameter and you want to make sure the broadcast goes to him also.

Below is an example from an AstaBDEServer.

```
6000:
  begin
    AstaServerSocket1.BroadCastProviderchanges(ClientSocket, nil,
  params[0].Name, 'customer', params[0].AsDataSet as TAstaDataSet,
  true);
  end;
6001:
  begin
    AstaServerSocket1.BroadCastProviderchanges(nil, nil, params[0].Name,
  'customer', params[0].AsDataSet as TAstaDataSet, true);
  end;
```

1.6.9.2 Automatic Provider Broadcasts

When using Provider broadcasts, AstaClientDataSets can automatically be updated with no code. To do this you must follow 3 rules.

1. The OnProviderBroadcast event cannot be assigned.
2. The ProviderOptions.BroadcastAction.baAuto must be set to True.
3. In order for #2 to function properly, primekey field information must be available on the client to do a locate (and with ASTA 3, to use FindKey).

In order to fetch primekey field information for any provider to allow the Provider.ProviderBroadcast.BroadcastAction to function as baAuto primekey field information must be available on the AstaClientDataSet. To retrieve primekey field information for providers, call the SetPrimeKeyFieldsFromProvider method after the Provider is opened and you are connected to the server. Use the OnAfterPopulate event to call this method.

```
if AstaClientDataSet1.PrimeFields.Count = 0 then
  AstaclientDataSet1.SetPrimeKeyFieldsFromProvider;
```

For a full discussion of Provider Broadcasts see:
http://www.astatech.com/support/white/conflictmanagement.asp

1.6.9.3 Provider Bite at the Apple

Warning: non-trivial discussion!

Broadcasts allow changes from from one user to be seen by other users, and optionally to have their TAstaClientDataSets automatically updated. Using SendProviderTransactions, multiple AstaClientDatasets can be updated on the server in one transaction.

ASTA 3 adds the ability to add manual broadcasts for providers. This allows you to create and fill a TAstaDataSet on the server with any kind of data you want and broadcast it out to users who have registered a provider for a broadcast. This broadcast can be initiated by a server side process, or by a message from a remote client to trigger the broadcast.

When AstaClientDataSets are used in conjunction with AstaProviders on the server, when ApplyUpdates is called, 2 Datasets are created, an OriginalValuesDataSet and a CurrentValuesDataSet. They are linked together by a Bookmark:Integer field so that on the server, the Provider Before/After events can be fired for each row of the OriginalValuesDataSet. You can allow the TAstaProvider to generate SQL or you can set the Handled:Boolean param to true and perform any database activity you choose.

We have had ASTA users, using Views, who use providers and take control of updating whatever tables they want, taking control of their own data destiny. But by doing this, they lose the ability to broadcast their changes since they have choosen to not allow ASTA to generate SQL for update.

ASTA 3 introduces a new Event and a couple of new Provider properties to allow for ASTA users to use Broadcasts even though they have elected to take control of the Update process in lieu of ASTA's default behavior.

The AstaServerSocket has an OnProviderCompleteTransaction event that will fire at the

end of the provider update process on the server when used from the client side call of SendProviderTransactions.

```
procedure OnProviderCompleteTransaction(Sender: TObject;
  TransactionCommitted: Boolean; const Providers: array of
  TAstaProvider);
```

All of the providers that participated in the SendProviderTransation call will be presented in this event as an Array of TAstaProvider.

In addition, there are some new TAstaProvider published properties:

```
property RefetchOnUpdates: Boolean;
//does refetches on updates as well as Inserts so that update trigger
  values can be refetched. used to be public in ASTA 2.6

property NoSQLFields: TStrings;
//allows fields to be removed from the SQL generation process. this
  supplements the existing logic to generate SQL for Readonly Fields.

property RetainCurrentValuesDataset: Boolean;
//This is the important one here. By setting this to true, the
  CurrentValuesDataSet will be available in the
  OnProviderCompleteTransaction Event so that it can be broadcasted to
  remote clients.
```

Below is an example of how to access the CurrentValuesDataSet in the OnProviderCompleteTransation Event:

```
procedure TServerDM.ServerSocketProviderCompleteTransaction(
  Sender: TObject; TransactionCommitted: Boolean; const Providers: array
  of TAstaProvider);
var
  i: Integer;
begin
  for i := Low(providers) to High(Providers) do begin
    if Providers[i].RetainCurrentValuesDataset then begin
      if providers[i].CurrentValuesDataSet = nil then
        Log(' nil ')
      else
        Log(Providers[i].Name + ' :' +
  IntToStr(Providers[i].CurrentValuesDataSet.RecordCount) + ' current
  values.');
    end else Log(Providers[i].Name);
  end;
end;
```

Thanks to our ASTA users in Scandinavian countries who seem to be big Provider Broadcast fans.

1.6.9.4 Caching Provider MetaData

When using ASTA providers and ServeMmethods and setting the ProviderName or ServerMethod Name at runtime for AstaClientDataSets, Param information must be retrieved from the server. In order to make this process more efficient, and to eliminate server round trips, Provider and ServerMethod information can be cached on the client.

The AstaClientSocket has a method that can fetch Provider and Servermethod information:

```
AstaClientSocket.CacheMetaDataOptions: TServerMetaCacheOptions
TServerMetaCacheOptions = (smNoCache, smFetchOnValidate,
  smLoadFromFile);
```

smNoCache	Does no MetaData caching.
smFetchOnValidate	This will retrieve fresh information from the ASTA server after a client has been properly authenticated.
smLoadFromFile	this will load the metadata cache from the filename identified by the CachMetaDataFileName property.

If you want to manually control this process use the following call after the client has been authenticated in the OnLoginParamsEvent:

```
AstaClientSocket.CacheproviderAndMethodInfoLocally(FileName: string;
  ForceServerRefresh: Boolean);
```

If your server will not change, you can store this information in a local file. Call it after your client has logged into the server.

Now anytime you set a provider or servermethod at runtime, the param information will be retrieved from a local store. In addition the Provider primekey field information will be stored in this cache so that support for the baAuto Broadcast Option will be transparent. It is recommended that MetaData be cached using the smFetchOnValidate property. This greatly reduces server round trips.

1.6.9.5 What Data is Sent in Provider Broadcasts

When using server side SQL, AstaClientDataSets do not generate SQL on the client but instead send the OldValuesDataSet and current values dataset only.

In order to preserve bandwidth, if blob and memo fields are not changed then the blob/memo fields are not sent but are instead set to null. This then makes it problematic as to how to set the blob/memo field in the OnProviderBroadcast event on the client.

1.6.10 Security Issues

Socket Addressses can be validated with the AstaServerSocket.OnClientValidate Event.

A Server Login can be required using the AstaServerSocket.OnClientLogin or OnClientDBLogin.

Remote Users who don't login correctly can be immediately disconnected using the AstaServerSocket.DisconnectClientsOnFailedLogin Property.

Design time Access can be controlled with DTUserName and DTPassword.

The AstaServer can set the public SecureServer:Boolean to true to ONLY allow for Login Messages to be processed to completely lockdown a server and force any access to require a Login before any other message can be processed.

ASTA Servers can be made very secure through a combination of

Authentication

Encryption

Defining a Back Door

Security Levels for users

By default, ASTA servers are shipped with no security to easily allow clients to connect to the server. Servers can be made secure in degrees and also allow for "back doors" to allow for administration and design time support. In addition, users can be allowed to access certain processes on servers.

Authentication

Using the OnClientAuthenticate event on the AstaserverSocket, users can be required to use a username and password. By default the AstaClientSocket will not even login to the server. To trigger the login process the AstaClientSocket.AutoLoginDlg must be set to anything but ltNoChallenge. When this happens UserName and Password along with the ClientsocketParams will be sent to the server after a client connects to the server. See the discussion on the Login process for more detail.

ASTA servers will allow for ASTA clients to successfully connect to the server with AutoLoginDlg set to ltNoChallenge. For new ASTA users, we have found that we must not secure servers too tightly as they will have difficulting connecting to servers.

To set an ASTA server so that the ONLY message that is allowed to be processed by any remote client is an Authentication message, containing the server name and password, set the AstaServerSocket.SecureSocket:Boolean to true.

Note: this will close the old design time back door used by DTAccess. Used TrustedAddressses if you want to allow for design time access with no login.

Socket Addressses can be validated with the AstaServerSocket.OnClientValidate Event.

Athentication occurs using the OnClientAuthentiate Event.

Remote Users who don't login correctly can be immediately disconnected using the AstaServerSocket.DisconnectclientOnFailedLogin Property.

Back doors can be defined with TrustedAddresses

(for backwards compatibility the old Design time Access is still available but using TrustedAddressess is now the recommended method of defining back doors).

The AstaServer can set the public SecureSocket :Boolean to true to ONLY allow for Login Messages to be processed to completely lockdown a server and force any access to require a Login before any other message can be processed.

Hardware can also be used to authenticate users allowing for only certain PC's, by registering the MAC address of the network card on the server, to be authenticated. SkyWire Clients (palm and Wince) also send unique identifiers to the server that can be used to insure that only authorized devices, with proper usename and password, that also use a recognized encryption scheme can connect to ASTA servers.

Defining a Back Door

the TAstaServerSocket has a TrustedAddresses:TStrings property that allows for IP Addresses to be defined that will be allowed to connect and process any kind of data on the server. If you set SecureServer to true and still want to allow for design time access to the server use the TrustedAddresses property to define what development machines can connect to your server.

Encryption
For production servers, encryption should be used in addition to Authentication. ASTA supplies native DES, AES and RSA Encryption that is available cross platform. Additionally, custom encryption can be used which allows for any Encyryption component to be used on ASTA servers and clients. The only reuqirement is that a passed in VAR string be encrypted or descrypted.

See Also

DES
AES
RSA Encryption

Security Levels for users
By default, ASTA servers support client side sql or server side processes. There are times when you do not want any access to SQL from remote clients or clients to be able to fetch MetaData information.

The OnBeforeProcessToken event allows any server side process to be limited to any or all users. In addition, the UserRecord.ParamList can be used to store any kind of information so you can code your server to have a security level for each client and check as to whether you want to permit access depending on some criteria and if not, raise an exception.

1.6.11 Sockets and TCP/IP Issues

Asta Transports
Asta and Blocking Sockets
Asta in an ISAPI DLL

Early Connects
Socket Errors
TCustomWinSocket

1.6.11.1 ASTA Transports

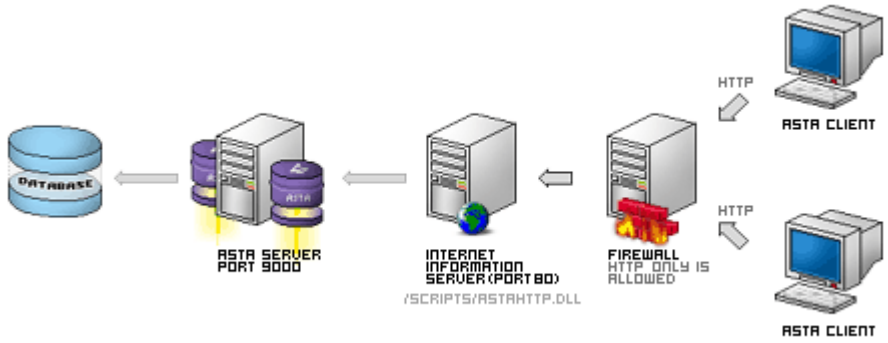
ASTA Clients can use a number of different protocols in communcation with remote ASTA Servers. The ASTA server is always aware of the protocol used by remote clients. So if a client is accessing a server from an ISAPI dll and using tcp/ip blocking calls or via WinInet using HTTP, the server will disconnect the client since they are "stateless". When using blocking calls the server must use Pooled or SmartThreading. ASTA Servers require no command line switches or protocol changes to support any ASTA client. One Server supports them all with the only requirement that they support Pooled Database Sessions. See the discussion on Threading ASTA servers for more detail.

TCP/IP with non-blocking sockets
This is the default mode of operation for ASTA clients. The TAstaclientSocket uses async calls so there is no need for any threading in client applications. Applications stay very responsive as the sockets are "event driven" so that the User Interface stays responsive.

TCP/IP with blocking calls
ASTA supports blocking calls for use in Isapi DLL's and in other cases where there is no windows message pump available. Blocking calls use the concept of TimeOut so the client will "block" for the Timeout: Integer (milliseconds) interval. When making a blocking call the server will "know" that the client is blocking and disconnect the client when any server side process is done. So if you set a large timeout value, the server will still disconnect the client as soon as possible. This is supported by the ASTA server threading calls from remote clients and, recognizing the client as stateless, forcing a disconnect at the end of any thread processing. Note: when running stateless use SendGetCodedParamLists calls if you want to use ASTA messaging. See Stateless Discussion.

SkyWire note: Traditional Palm and WinCE clients will use the same blocking calls. The server will recognize that they are palm or WinCE clients and disconnect them also UNLESS the SetTerminateConnection call is set to NOT terminate the client.

HTTP clients
In order to traverse or "tunnel" through firewalls, ASTA supports stateless HTTP clients that masquerade their traffic as HTTP. This is accomplished by adding HTTP header information to the ASTA binary transport.



HTTP clients require the use of a Web Server like Microsoft Internet Information Server (IIS), Apache or the Omni Server (www.omnicron.ca). ASTA comes with an isapi DLL (Astahttp.dll) that is placed in the scripts directory or equivalent of IIS and receives requests from ASTA clients and relays them to an ASTA server. See the TAstaClientSocket.WebServer property for a discussion on how to setup ASTA clients to use http tunneling. Using ASTA WinINet support is the preferred method to guarantee that any firewall can be traversed as WinInet uses the same settings as Internet Explorer.

SkyWire Note: SkyWire clients also support http tunneling which is required for Palm VII support via Palm.NET

1.6.11.2 Asta and Blocking Sockets

Since ASTA uses Non-blocking and event driven sockets which require windows messaging using ASTA Client components in an ISAPI dll has been problematic and we have supplied AstaWinManagement.pas in order to add a message pump to an ISAPI DLL but the results have not been consistent.

So we have added some calls to allow the ASTA client socket to be used in blocking mode. To allow the ASTAClientDataSet to be used under ASTA 2.5 we have added routines to "cache" dataset calls if the AstaClientSocket.ClientType is ctBlocking.

This means that once the AstaclientSocket is set to block you can call normalAstaclientDataSet calls and they will be cached until you say:

```
AstaClientSocket.SendAndBlock(100);
```

Where 100 is a timeout value of how long to wait (see TWinSocketStream for details on this in the Delphi Help). Any error is returned as the function result. If you want to raise an exception on the same call use:

```
AstaClientSocket.SendAndBlockException(100); //An exception will be
  raised
```

The following are examples of how to use these new calls:

```
procedure TWebModule1.WebModule1WebActionItem1Action(Sender: TObject;
  Request: TWebRequest; Response: TWebResponse; var Handled: Boolean);
```

```
begin
  try
    AstaclientDataSet1.OpenWithBlockingsocket(500, True);
    Response.Content := '</HTML> RecordCount is ' +
  IntTostr(AstaClientDataSet1.RecordCount) + '  </HTML>';
  except
    Response.Content := '<HTML>' + Exception(ExceptObject).Message +
  '</HTML>';
  end;
  Handled := True;
end;

procedure TWebModule1.WebModule1WebActionItem2Action(Sender: TObject;
  Request: TWebRequest; Response: TWebResponse; var Handled: Boolean);
begin
  AstaClientSocket1.ClientType := ctBlocking;
  // you can now make as many calls as you want and they will be cached
  from the client
  AstaClientDataSet1.ExecSQLString('UPDATE Customer SET Company = ' +
  QuotedStr(DateTimeToStr(now)) + 'WHERE custno = 1221');
  //they will all be sent when you send this
  try
    AstaClientSocket1.SendAndBlockException(100);
    Response.Content:='</HTML> Update Done </HTML>';
  except
    Response.Content:='<HTML>' + Exception(ExceptObject).Message +
  '</HTML>';
  end;
  Handled := True;
end;
```

1.6.11.3 ASTA in an ISAPI DLL

Since ASTA uses non-blocking and event driven sockets which require windows messaging, using ASTA client components in an ISAPI DLL has been problematic. We have supplied AstaWinManagement.pas in order to add a message pump to an ISAPI DLL, but the results have not been consistent.

We have added some new calls to allow the ASTA client socket to be used in blocking mode. To allow the ASTAClientDataSet to be used, under ASTA 2.5 we have added routines to "cache" dataset calls if the AstaClientSocket.ClientType is ctBlocking. This means that once the AstaClientSocket is set to block you can call normal AstaclientDataSet calls and they will be cached until you say AstaclientSocket1.SendAndBlock(100). Where 100 is a timeout value of how long to wait (see TWinSocketStream for details on this in the Delphi Help).

The following are examples of how to use these new calls.

```
procedure TWebModule1.WebModule1WebActionItem1Action(Sender: TObject;
  Request: TWebRequest; Response: TWebResponse; var Handled: Boolean);
begin
  AstaclientDataSet1.OpenWithBlockingSocket(500);
  Response.Content := '</HTML> RecordCount is ' +
  IntToStr(AstaClientDataSet1.RecordCount) + ' </HTML>';
  Handled := True;
```

```
end;

procedure TWebModule1.WebModule1WebActionItem2Action(Sender: TObject;
  Request: TWebRequest; Response: TWebResponse; var Handled: Boolean);
begin
  AstaClientSocket1.ClientType := ctBlocking;
  //You can now make as many calls as you want and they
  // will be cached from the client
  AstaClientDataSet1.ExecSQLString('UPDATE Customer SET Company = ' +
  QuotedStr(DateTimeToStr(Now)) + ' WHERE CustNo = 1221');
  //They will all be sent when you send this
  AstaClientSocket1.SendAndBlock(100);
  Response.Content := '</HTML> Update Done </HTML>';
  Handled := True;
end;
```

1.6.11.4 Early Connect

The earliest you can access an Asta server is in the OnConnect of the AstaClientSocket. The formcreate or formshow will most probably be called before the AstaclientSocket connects to the server.

ASTA uses non-blocking event driven Sockets which are asyncronous by nature. You can't write procedural code. We have shielded you from this everywhere but in the Connect.

Move your intialization code to the OnConnect method of the AstaClientSocket as that is where you know the socket will be connected.

Some people do this but coding the OnConnect event is preferable. ASTA has a method OpenTheSocket that does exactly this.

```
AstaClientSocket1.Active := True;
while not Astaclientsocket1.Active do
  Application.ProcessMessages;
```

When you have an AstaClientDataSet open at design time here is what we do behind the scenes.

1. The AstaclientDataSet will be opened in the formcreate. The socket is not connected yet so ASTA throws it in a List and closes it.
2. After the AstaclientSocket connects ASTA iterates through all the AstaclientDataSets found in step #1 and opens them again.
3. At run time when you say AstaClientDataSet.Open ASTA goes into a wait state using AstaSmartWait to allow you to write procedural code and to shield you from the true asyncronous nature of sockets.

Note: Asta messaging is asyncronous also. You do a SendCodedMessage and then handle it in the OnCodedMesssage. However, we added a SendGetCodedParamList that uses AstaSmartWait to allow you to write procedural code with Asta Messaging also.

The most Secure time to fetch data from an ASTA server is NOT in the OnConnect but after a successful OnLoginAttempt.

1.6.11.5 Socket Errors

Windows Sockets Code	Error	Description
WSAEINTR	10004	Interrupted system call.
WSAEBADF	10009	Bad file number.
WSEACCES	10013	Permission denied.
WSAEFAULT	10014	Bad address.
WSAEINVAL	10022	Invalid argument.
WSAEMFILE	10024	Too many open files.
WSAEWOULDBLOCK	10035	Operation would block.
WSAEINPROGRESS	10036	Operation now in progress. This error is returned if any Windows Sockets API function is called while a blocking function is in progress.
WSAEALREADY	10037	Operation already in progress.
WSAENOTSOCK	10038	Socket operation on nonsocket.
WSAEDESTADDRREQ	10039	Destination address required.
WSAEMSGSIZE	10040	Message too long.
WSAEPROTOTYPE	10041	Protocol wrong type for socket.
WSAENOPROTOOPT	10042	Protocol not available.
WSAEPROTONOSUPPORT	10043	Protocol not supported.
WSAESOCKTNOSUPPORT	10044	Socket type not supported.
WSAEOPNOTSUPP	10045	Operation not supported on socket.
WSAEPFNOSUPPORT	10046	Protocol family not supported.
WSAEAFNOSUPPORT	10047	Address family not supported by protocol family.
WSAEADDRINUSE	10048	Address already in use.
WSAEADDRNOTAVAIL	10049	Cannot assign requested address.
WSAENETDOWN	10050	Network is down. This error may be reported at any time if the Windows Sockets implementation detects an underlying failure.
WSAENETUNREACH	10051	Network is unreachable.
WSAENETRESET	10052	Network dropped connection on reset.
WSAECONNABORTED	10053	Software caused connection abort.
WSAECONNRESET	10054	Connection reset by peer.
WSAENOBUFS	10055	No buffer space available.
WSAEISCONN	10056	Socket is already connected.
WSAENOTCONN	10057	Socket is not connected.
WSAESHUTDOWN	10058	Cannot send after socket shutdown.

1.6.11.6 TCustomWinSocket

TCustomWinSocket is the base class for all Windows socket objects in the Borland Sockets which ASTA uses. See the Delphi help for more detail. This is a pointer pointing to the the actual ClientSocket of any remote clients. A TCustomWinSocket is required to communicate with remote clients and is maintained in the AstaServerSocket.UserList.

Unit
Scktcomp

Description
TCustomWinSocket introduces properties, events, and methods to describe an endpoint in a Windows socket connection. Descendants of TCustomWinSocket are used by socket components to manage the Windows socket API calls and to store information about a socket communication link.

A Windows socket encapsulates a set of communication protocols to allow the application to connect to other machines for reading or writing information. Windows sockets provide connections based on the TCP/IP protocol. They also allow connections that use the Xerox Network System (XNS), Digital's DECnet protocol, or Novell's IPX/SPX family. Sockets allow an application to form connections to other machines without being concerned with the details of the actual networking software.

Descendants of TCustomWinSocket represent different endpoints in socket connections. TClientWinSocket represents the client endpoint of a socket communication link to a server socket. TServerWinSocket represents the server endpoint of a listening connection. TServerClientWinSocket represents the server endpoint of a socket communication link to a client socket.

1.6.12 XML Support

ASTA 3 XML support is provided at 3 levels

1. Native DOM Parser and Thunk Layer (source code users only)
2. AstaDataSet XML Support
3. AstaParamList XML Support

ASTA 3 contains a native DOM XML parser implemented in pascal and also a "thunk" layer to use the Microsoft XML parser. The thunk layer can only be used by ASTA source code users and affects DataSet routines only.

Modify the defines in XML_Thunk.pas to link in the ASTA native parser, Delphi 6 XML helpers or the MS Parser.

```
{$IFDEF USE_LEGAL_D6}
  xmldom, XMLDoc, XMLIntf
{$ELSE} //end USE_LEGAL_D6
{$IFDEF   USE_MSXML}
  ActiveX, MSXML_TLB
{$ELSE} //end USE_MSXML
  astaxml_dom, astaxml_xmlv, astaxml_xmlw
{$ENDIF   USE_MSXML}
{$ENDIF   USE_LEGAL_D6}
```

ASTA Datasets and ParamLists can load and save XML files. The AstaXMLExplorer tutorial

shows how to use the ASTA DOM XML parser to load and browse the XML attributes of any XML File.AstaDatasets can save and load to XML using the ADO or DataSnap (midas) Formats.

The TAstaDataset has several methods for XML support and supports both the ADO and Midas format.

```
TAstaXMLDataSetFormat = (taxADO, taxMidas);
procedure LoadFromXML(const FileName: string);
procedure LoadFromXML(Const Stream:Tstream);
//Loads from an XML file or Stream having either the Midas or ADO
  format.

procedure SaveToXML(const FileName: string; XMLFormat:
  TAstaXMLDataSetFormat); overload;
//Saves to an XML file having the Midas or ADO Format.

procedure SaveToXML(Stream: TStream; XMLFormat: TAstaXMLDataSetFormat);
  overload;
//Saves to a stream having a midas or ado format.
```

TAstaParamLists can be saved and loaded from XML also as implemented in AstaParamListXML.pas

ParamList -> XML routines

```
procedure ParamListToXML(const Params: TAstaParamList; Doc: TXMLDocument);
function ParamListToXMLdoc(const Params: TAstaParamList): TXMLDocument;
function ParamListToXMLstring(const Params: TAstaParamList): string;
procedure ParamListToXMLstream(const Params: TAstaParamList; Stream:
  TStream);
```

XML -> ParamList routines

```
procedure XMLToParamList(const Doc: TXMLDocument; Params: TAstaParamList);
function XMLdocToParamList(const Doc: TXMLDocument): TAstaParamList;
function XMLstringToParamList(const S: string): TAstaParamList;
procedure XMLstreamToParamList(Stream: TStream; Params: TAstaParamList);
```

See the ASTA 3 Tutorials
Asta3XMLDataSet
XMLExplorer

The ASTA 2 XML routines are maintained for backwards compatibility but we recommend you move to the new DOM XML Parser that use standard formats.

```
procedure DataSetToXMLStream(D: TDataSet; TableName: string; var MS:
  TStream; Blobs, Memos, MetaData, DTD, AttributeData: Boolean);

function DataSetToXMLStreamF(D: TDataSet; TableName: string; Blobs,
  Memos, MetaData, DTD, AttributeData: Boolean): TMemoryStream;

procedure DataSetLoadFromXML(D: TAstaDataSet; FileName: string);

procedure DataSetLoadFromXML(D: TAstaDataSet; MS: TMemoryStream);
```

1.6.13 Cross Platform Support

1.7 Developing for the Internet

It used to be that all Business Applications were Database Applications. Now with the Internet, all business applications are Database Applications with an Internet Strategy. Most companies support desktop browser clients whether they are customers making e-commerce purchases or employees taking care of business.

The second stage of the Internet is upon us. The first stage secured the browser as an integral part of normal business practices. The second stage will open up Business-to-Business channels through Web Services, as Information Systems inter-communicate. Mobile users with PDAs, PDA-Phones and Tablets will demand support beyond the browser.

In this document, we will discuss:

- The evolution of Database Application development
- How the Internet has changed the concept of application development
- Issues and methods related to developing for the Internet
- Supporting more than Windows Desktops users via a Browser

1.7.1 Database Application Development

File Server, Client/Server, Application Server

1.7.1.1 File Server

Database application development started with File Server databases, often referred to as "Desktop Databases". Examples of these include dBase, Paradox and Access. These databases were just files that each user accessed directly over a network or from a local machine. Applications contained all the "business Logic" in the EXE itself and when accessing the database "files," portions of the file itself and associated indexes were transported over the wire. Typically, there were file corruption issues, and when any new application was created, it needed to contain all the business logic in the program itself. A proprietary API (Application Programming Interface) was used to access the files. There was no security built into the system.

Table 1 outlines design aspects of the file server model.

Table 1: FILE SERVER OVERVIEW

SYSTEM FEATURES	METHODS / PERFORMANCE
Scalability	Limited by File system
Business Logic	Resident in application EXE
Performance	Prone to file corruption
Access	Each database required own API
Security	None
Deployment	Network Read/Write access to file system
Transport	Operating System File Access Calls not encrypted
State	Maintains State

1.7.1.2 Client Server

In the 1980's Client/Server systems started to appear. Clients no longer directly accessed "files" over the network, but communicated with a "Server". Result sets or "copies" of the data were transported over the wire and SQL became the new standard API to access the Database. Client/Server Databases now also allowed for the "business Logic" to be stored in the database itself through the use of foreign keys, domains, stored procedures and triggers. The "Business Logic" could now be split between a FAT client and the server. Client applications were said to be FAT as they often required extensive setup programs and a number of DLL's to be installed. Each client used their own user name/password to connect to the database as Client Server introduced security features.

Figure 1 shows basic client server design; Table 2 covers design aspects.

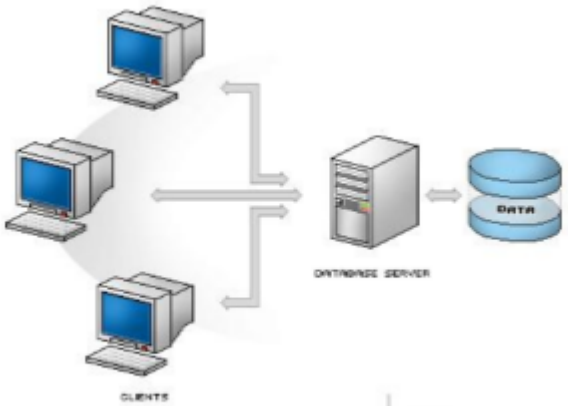


Fig 1
Client-Server Application

Table 2: CLIENT SERVER OVERVIEW

SYSTEM FEATURES	METHODS/PERFORMANCE
Scalability	Result Sets allow for more users than File Server Systems. Each user requires a database connection.
Business Logic	In the Database and Client EXE
Performance	Moves only result sets rather than files
Access	SQL
Security	Database Security Features
Deployment	Fat Clients require Database DLL's and setup
Transport	Named Pipes, TCP/IP not encrypted
State	Maintains State

1.7.1.3 Internet and Web Development

The mid 1990's brought the Internet blowing through the world and changing it forever. The Internet introduced the "N Tier" model with the benefits of scalability, cross platform support and thin clients.

Figure 2 represents the basic n-tier model; Table 3 highlights design details of N-Tier

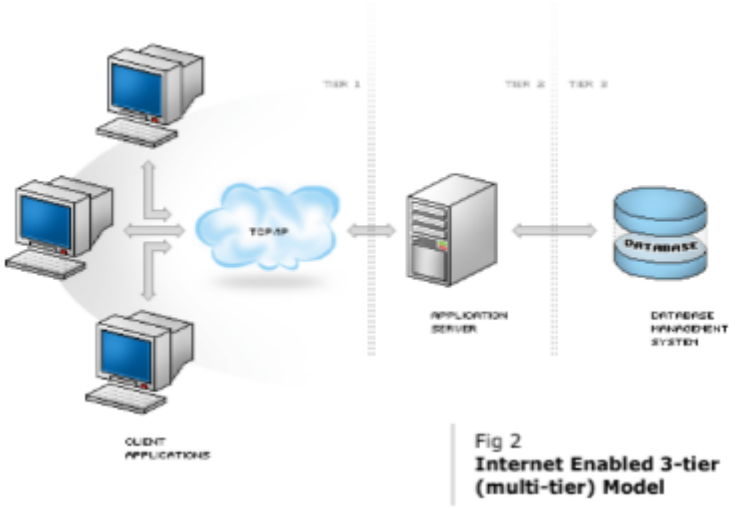


Fig 2
Internet Enabled 3-tier (multi-tier) Model

Table 3: WEB SERVER AND "N-TIER" OVERVIEW

SYSTEM FEATURES	METHODS/PERFORMANCE
Scalability	Scalability inherent in the design. Just add more bandwidth and servers. Stateless HTTP protocol adds to scalability.
Business Logic	Business logic could be split between the client/server database used on the Web Server and the HTML created on the web server.
Performance	One web server supports many browser clients. Database connections can be "pooled" or shared.
Access	Virtually unlimited authorized browser clients.
Security	There is some security inherent in using a Web Server, if someone hacked the web server they would have direct access to the database.
Deployment	Thin Client, Cross Platform extends capacity, No DLL's are required on the client. Anyone using a browser can access the server.
Transport	HTTP
State	Stateless

1.7.2 Web Development and Application Servers

As Web development matured, the use of an Application Server became "best practice". Both Java and Microsoft (MTS) offered application Server choices. By combining the best of client/server databases and Application Servers, modern developers can build secure, scalable applications. Instead of having web applications access the Database directly, Remote Procedures are coded on the Application server. Web Developers no longer need to know which table or even what database is in use. Application level architects code the business logic on the application Server so that any application uses the same business rules. Application Developers can concentrate on delivering the application rather than database specific issues.

Using an Application Server also opens up the ability for Businesses to communicate with each other using Web Services or SOAP, as Remote Procedures are made available to business partners.

Table 4 covers the same aspects of earlier database server designs

Table 4: APPLICATION SERVER OVERVIEW

SYSTEM FEATURES	METHODS/PERFORMANCE
Scalability	Database connection pooling handled by the Application Server.
Business Logic	The application server could be a repository of "business Logic" in one central place. Business partners can use "Remote Procedures" additionally as SOAP Services.
Performance	Improved through the use of the Application Server.
Access	Virtually unlimited authorized browser clients.
Security	Adds a layer of security, as now the web server cannot access the database directly. Creates a DMZ (Demilitarized Zone). SSL over HTTP.
Deployment	Thin Client, Cross Platform extends capacity, No DLL's are required on the client. Anyone using a browser could access the server.
Transport	HTTP
State	Stateless

1.7.3 Beyond the Browser

We have discussed traditional web development and shown how using an Application server provides functionality to ensure security, scalability and the centralization of business logic. All this, of course, is for traditional web development using html and http. No one can dispute the popularity of browser applications. It should be noted however, that the two most popular applications that run on the Internet - Email and Instant Messaging - are not browser applications, but normal Windows applications.

Windows users are accustomed to features like MDI and fancy reports when running their normal business applications. The browser interface is quite limited compared to the Rich Thin Client applications that ASTA can provide. If the Internet is one big network, and everybody is connected why not allow them to use Windows as it was intended? Why not stay connected to the server and use TCP/IP so that the server can "push" out information. Why not allow clients to communicate directly with each other?

Figure 3 depicts a basic ASTA N-Tier Application; Table 5 shows enhancements.

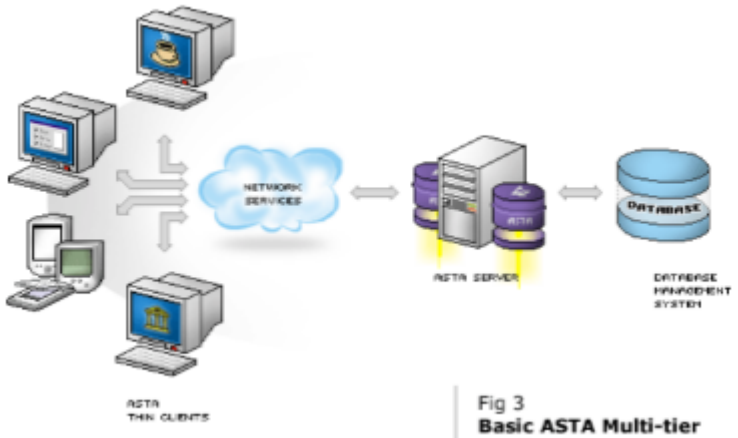


Fig 3
Basic ASTA Multi-tier Application

Table 5: ASTA APPLICATION SERVER OVERVIEW

SYSTEM FEATURES	METHODS/PERFORMANCE
Scalability	Database connection pooling and all thread issues handled by the easy to deploy Application Server. Load Balancer available.
Business Logic	Supports ServerMethods, AstaProviders, and Messaging Layer. ServerMethods can be "published" as SOAP Services.
Performance	Improved through the use of the application Server. Cross Platform compression enhances performance.
Access	Virtually unlimited authorized browser clients. Win32 Clients. Kylix Clients. Palm, WinCE, Java, GCC Linux, C#, and embedded Chips. Cross Platform C++ Clients as well as Java Clients.
Security	DES, AES, or RSA Encryption as well as SSL via WinInet.
Deployment	Committed to Thin Client deployment with AutoUpdate feature cross platform.
Transport	HTTP, TCP/IP, UDP
State	Stateless or Stateful

1.7.4 Additional Tables and Features

ASTA was conceived in 1997 in order to integrate the Internet with traditional client/server database applications. ASTA Applications are Thin Client Internet database applications that open up worlds beyond traditional database - and even Web - development. With scalable cross platform database application servers, and cross platform clients that not only support database applications but collaborative Internet applications as well, the ASTA SkyWire framework provides an all-encompassing fabric. Using this fabric, any database application with an Internet Strategy can support other clients in addition to traditional desktop browsers, and also add collaborative abilities and the capacity to work disconnected. Desktop users are going mobile and want the same information outside the office they are accustomed to on the desktop. Browser users of the world unite! You have nothing to lose but your Ethernet cables!

As the Internet moves into the Second Wave, and more users go mobile, Application developers must reach into a toolbox that can solve problems that go well beyond the traditional database toolset. "There is more to heaven and earth, Horatio, than Browser applications."

The remaining pages contain tables and figures illustrating the feature rich and flexible capacities of the ASTA technologies.

1.7.4.1 ASTA Framework

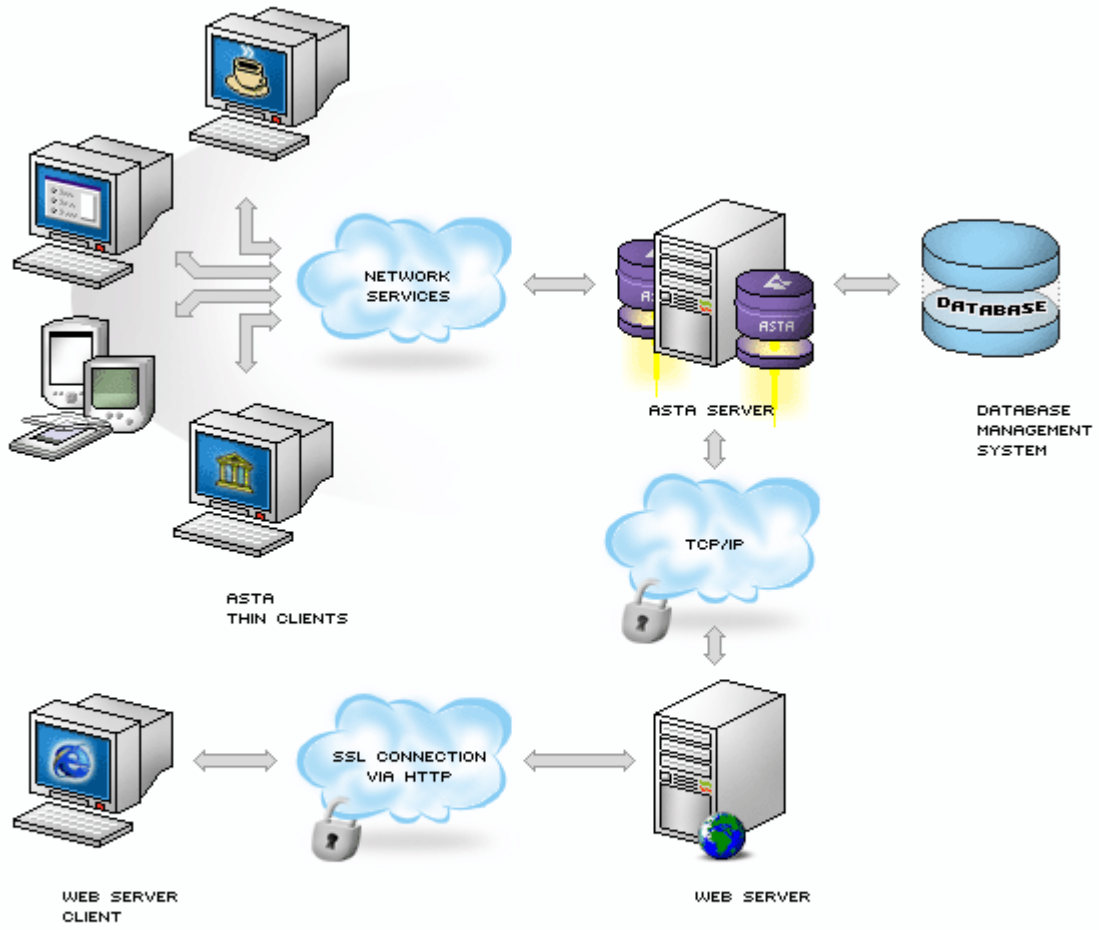
Table 6 highlights some of ASTA's Design Features:

Table 6: ASTA Framework

Feature	Description
Cross Platform Messaging	Full Cross Platform Messaging Layer for any datatypes.
Security	Cross Platform DES, AES, RSA, SSL via WinINet or custom encryption options.
Instant Messaging API	Formal Instant Messaging API to make building IM apps very easy.
File Synchronization	Win32 and WinCE File Synchronization Technology. Available as Namespace extension to run from Windows Explorer.
Database Synchronization	SkySync for cross platform Database Synchronization.
Napster Like File Sharing	ASTA Napster like API to allow for efficiently searching and secure streaming of files from collaborative workgroups.
Auto Client Update	New Versions of client app registered on server and streamed down as appropriate.
Super Suitcase Model for Disconnected Users	Virtual DataSets with native SQL support for low memory use and disconnected Application development.
Voice over IP	Wince and WinCE.
Cached Updates on All Platforms	ClientDataSet features supported Cross Platform.
SOAP Support	ServerMethods can be published as SOAP Services. Cross Platform SOAP Clients.
Charting Tools	Ability to define graphs via thin client ASTA enabled apps with the ability to tie the graph to SQL Queries. Palm and WinCE clients with drill down.
Thin Client ODBC Driver	ASTA ODBC Driver to use with Legacy 2 tier Applications.
JDBC Driver	Full Java JDBC Driver.
EJB Access from Delphi and Beyond	JAsta allows for EJB's to be created and EJB methods executed from ASTA Cross Platform Clients.
Video Streaming	Cross Platform Video Streaming
Video Conferencing	H323 Video Conferencing

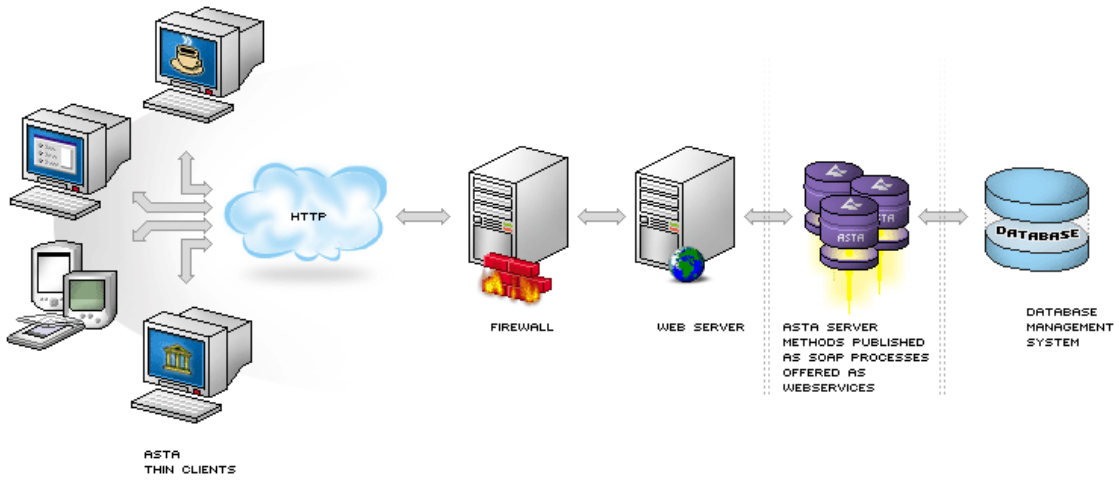
1.7.4.2 TCP/IP and HTTP Clients

Figure 4: ASTA Supports TCP/IP as well as HTTP Clients



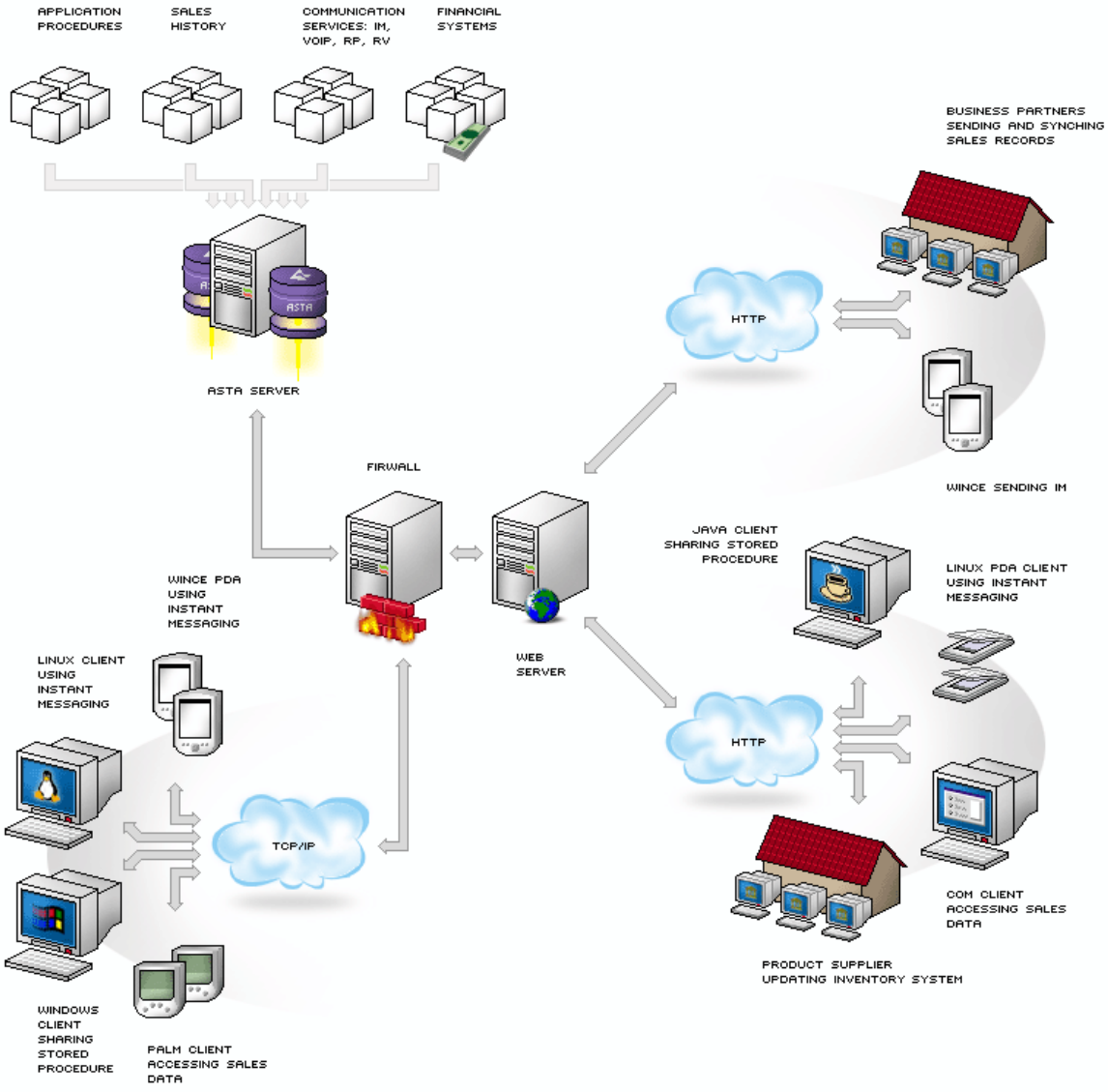
1.7.4.3 ServerMethods as SOAP Services

Figure 5: ASTA ServerMethods can be published as SOAP Services



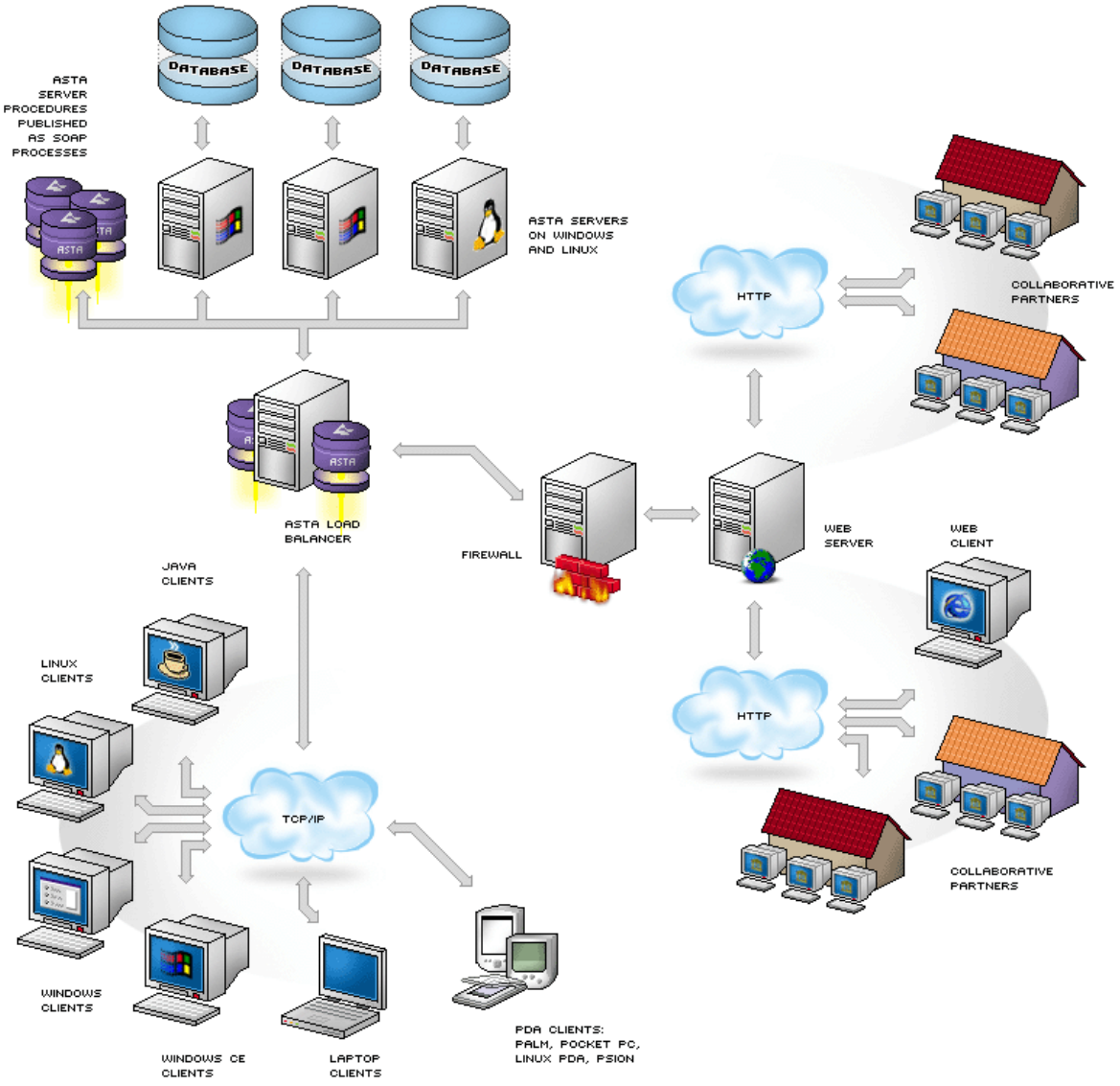
1.7.4.4 Business to Business

Figure 6: ASTA supports Business to Business



1.7.4.5 Cross Platform/Device/Protocol

Figure 7: ASTA Supports Cross-Platform, Cross-Device, Cross Protocol Design



1.8 ASTA Components

Visual Key to the components

AstaClientDataSet

The AstaClientDataSet is the key component in the architecture. It descends from the VCL's TDataSet object. In essence, the AstaClientDataSet is a hybrid TTable-TQuery-TUpdateSQL component that resides in memory. Since it is loaded into memory, access to records in the AstaClientDataSet is extremely fast. If you need persistence, the dataset can be streamed to a file and then loaded from the file. You point a TDataSource component to the AstaClientDataSet in the same way that you would point a

TDataSource to a TTable or TQuery.

 AstaClientSocket

The AstaClientSocket is the component that connects the client to the AstaServer or middle-tier. It descends from a combination of the VCL's socket objects. The AstaClientSocket is the client end of the "pipe". All data sent to or from the application passes through this component. The AstaClientSocket's Address and Port properties need to match the server's Address and Port properties (the Host and Port properties can be used if you are using the Domain Naming System [DNS]).

The AstaClientSocket, is also a place where "global" characteristics can be assigned to the client piece of the application. If you wish to use compression or encryption, for instance, you would set those at the AstaClientSocket.

 AstaServerSocket

The AstaServerSocket is the AstaServer component that accepts the connections from the AstaClients. It descends from the VCL's socket objects. The AstaServerSocket is the center control of the application server. It manages who is connected and which messages they are receiving. Every message in an AstaServer flows through the AstaServerSocket.

Asta Clients
AstaClients are normally comprised of an AstaClientSocket and at least one AstaClientDataSet. The AstaClientSocket is in charge of "speaking" to the AstaServer or, more specifically, to the AstaServerSocket.

The AstaClientDataSet is the TDataSet descendant. It is an in-memory dataset that frees you from the BDE. The AstaClientDataSet functions like a hybrid TTable-TQuery-TUpdateSQL component. You point a TDataSource to the AstaClientDataSet and then you connect your normal data-aware controls to the TDataSource.

Asta Servers
An AstaServerSocket is the core component of an ASTA server. The ASTA server is the "application server" or "middle tier". It manages all the client connections and simultaneously speaks to the database (via regular database controls) as necessary. As depicted in the diagram, the ASTA server easily speaks to BDE or ODBC data sources. Although not depicted in the diagram, the ASTA server can be made to speak to other data sources as well. In fact, if you are willing to write an interface, the ASTA server can be hooked up to anything you can think of. Not just databases, but process control or custom-made equipment.

1.8.1 Client Side

1.8.1.1 TAstaDataset

Properties : Methods : Events

Unit
AstaDrv2

Declaration
```
type TAstaDataSet = class(TAstaCustomDataSet);
```

Description
The TAstaDataSet set is an in-memory dataset. It descends from the TDataSet object in the Delphi VCL. The TAstaDataSet can be used like a Delphi TTable but it doesn't use the BDE. In essence, it is the "thin" in "thin client". Since it resides in memory, you will also find that it is very fast.

The TAstaDataSet implements master-detail support, indexing, persistence, cloning, streaming, xml and filtering.

The same methods and techniques that you can use with TTables are used with the TAstaDataSet. Be certain to define fields before using the TAstaDataSet.

See also: Functions for Working With TAstaDataSets

1.8.1.1.1 Properties

TAstaDataSet

Aggregates
FieldsDefine
Indexes
IndexFieldCount
IndexFieldNames
IndexFields
IndexName
KeyExclusive
KeyFieldCount
MasterFields
MasterSource
ReadOnly
StreamOptions

Derived from TDataSet
Active
AggFields
AutoCalcFields
BlockReadSize
Bof
Bookmark
CanModify
Constraints
DataSetField
DataSource
DefaultFields
Designer
Eof
FieldCount
FieldDefList
FieldDefs
FieldList

Fields
FieldValues
Filter
Filtered
FilterOptions
Found
Modified
ObjectView
RecordCount
RecNo
RecordSize
SparseArrays
State

1.8.1.1.1.1 TAstaDataSet.Aggregates

TAstaDataSet

Declaration
```
property Aggregates: TAstaAggregate;
```

Description
Use Aggregates to define aggregates that summarize the data in the client dataset. Aggregates is a collection of TAstaAggregate objects, each of which defines a formula for calculating an aggregate value from a group of records in the client dataset. The individual aggregates can summarize all the records in the client dataset or subgroups of records that have the same value on a set of fields. Aggregates that summarize a subgroup of records are associated with indexes, and can only be used when the associated index is current.

TAstaAggregates is the type of the Aggregates property, which represents all the maintained aggregates for a client dataset. A single TAstaAggregate object represents each maintained aggregate. Maintained aggregates summarize data over the records in the client dataset.

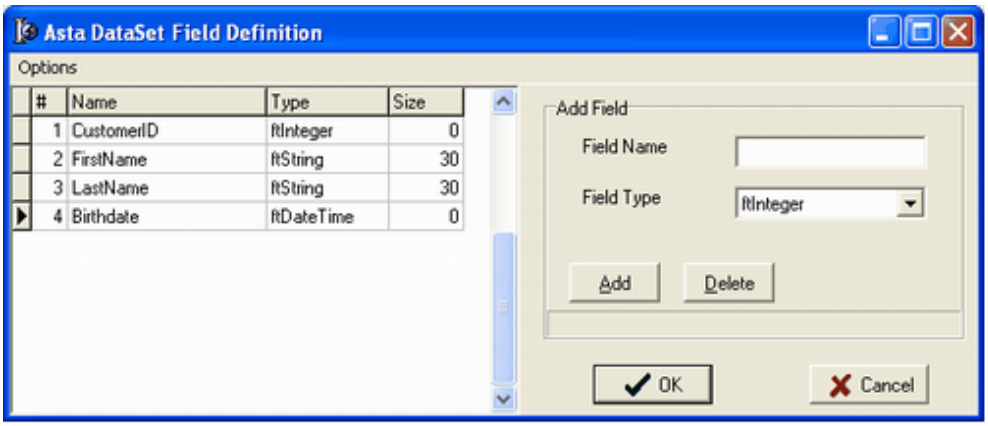
TAstaAggregates can include a mix of active and inactive aggregate objects. Active aggregates are updated dynamically as the data in the client dataset is edited. Inactive aggregates define a formula for summarizing data and a group of records to summarize, but these formulas are not evaluated. Aggregates can be inactive because they are not currently needed, or because the index that defines the group of records they summarize is inactive.

1.8.1.1.1.2 TAstaDataset.FieldsDefine

TAstaDataSet

Declaration
```
property FieldsDefine;
```

Description
The FieldsDefine property editor allows you to create fields for the AstaDataSet. Input the name, type, and size if applicable.



1.8.1.1.1.3 TAstaDataset.Indexes

TAstaDataSet

Declaration
```
property Indexes: TAstaIndexes;
```

Description
Asta datasets can have indexes defined so that multiple sort orders can be defined. Indexes can be added at design time using the Indexes property or at runtime by calling AddIndex or AddIndexFields. The following properties and methods are used in conjunction with indexes:

Properties
IndexFieldCount
IndexFields
KeyExclusive
KeyFieldCount

Methods
ApplyRange
CancelRange
EditRangeEnd
EditRangeStart
SetRange
SetRangeEnd
SetRangeStart
SetKey
EditKey
FindKey
FindNearest
GotoKey
GotoNearest
AddIndexFields
AddIndex

1.8.1.1.1.4 TAstaDataSet.IndexFieldCount

TAstaDataSet

Declaration
```
property IndexFieldCount: Integer;
```

Description
Check IndexFieldCount to determine how many fields are used by the current index for the dataset.

1.8.1.1.1.5 TAstaDataSet.IndexFieldNames

TAstaDataSet

Declaration
```
property IndexFieldNames: string;
```

Description
Use IndexFieldNames as an alternative method of specifying the index to use for a client dataset. Specify the name of each field on which to index the dataset, separating names with semicolons. Ordering of field names is significant.

Indexes added using IndexFieldNames do not support grouping or maintained aggregates.

Note: The IndexFieldNames and IndexName properties are mutually exclusive. Setting one clears the other.

1.8.1.1.1.6 TAstaDataSet.IndexFields

TAstaDataSet

Declaration
```
property IndexFields[Index: Integer]: TField;
```

Description
IndexFields is a zero-based array of field objects, each of which corresponds to a field in the current index. Index is an ordinal value indicating the position of a field in the index. The first field in the index is IndexFields[0], the second is IndexFields[1], and so on.

Note: Do not set IndexField directly. Instead use the IndexFieldNames property to order datasets on the fly at runtime.

1.8.1.1.1.7 TAstaDataSet.IndexName

TAstaDataSet

Declaration
```
property IndexName: string;
```

Description
Use IndexName to specify an alternative index for a client dataset. If IndexName contains a valid index name, then that index is used to determine sort order of records.

Note: IndexFieldNames and IndexName are mutually exclusive. Setting one clears the other.

1.8.1.1.1.8 TAstaDataSet.KeyExclusive

TAstaDataSet

Declaration
```
property KeyExclusive: Boolean;
```

Description
Use KeyExclusive to specify whether a range includes or excludes the records that match its specified starting and ending values. By default, KeyExclusive is False, meaning that matching values are included.

To restrict a range to those records that are greater than the specified starting value and less than the specified ending value, set KeyExclusive to True.

1.8.1.1.1.9 TAstaDataSet.KeyFieldCount

TAstaDataSet

Declaration
```
property KeyFieldCount: Integer;
```

Description
Use KeyFieldCount to limit a search based on a multi-field key to a consecutive subset of the fields in that key. For example, if the primary key for a dataset consists of three-fields, a partial-key search can be conducted using only the first field in the key by setting KeyFieldCount to 1. Setting KeyFieldCount to 0 allows the client dataset to search on all key fields.

Note: Searches are only conducted based on consecutive key fields beginning with the first field in the key. For example if a key consists of three fields, an application can set KeyFieldCount to 1 to search on the first field, 2 to search on the first and second fields, or 3 to search on all fields. By default KeyFieldCount is initially set to include all fields in a search.

1.8.1.1.1.10 TAstaDataSet.MasterFields

TAstaDataSet

Declaration
```
property MasterFields: string;
```

Description
The MasterFields property is used for setting up master/detail relationships. This property must be used in conjunction with the MasterSource property and the detail table needs to use a parameterized query. A master/detail relationship can also be used in Suitcase mode using filters when the TAstaClientDataSet is disconnected from the server.

1.8.1.1.1.11 TAstaDataSet.MasterSource

TAstaDataSet

Declaration
```
property MasterSource: string;
```

Description
The MasterSource property determines the master table in a master/detail relationship. The property must be used in conjunction with the MasterFields property

1.8.1.1.1.12 TAstaDataSet.ReadOnly

TAstaDataSet

Declaration
```
property ReadOnly: Boolean;
```

Description
Use the ReadOnly property to prevent users from updating, inserting, or deleting data in the dataset. By default, ReadOnly is False, meaning users can potentially alter the dataset's data.

To guarantee that users cannot modify or add data to a dataset, set ReadOnly to True.

1.8.1.1.1.13 TAstaDataSet.StreamOptions

TAstaDataSet

Declaration
```
property StreamOptions: TAstaStreamSaveOptions;

TAstaStreamSaveOptions = set of TAstaStreamSaveOption;
TAstaStreamSaveOption = (ssAstaLegacy, ssZlibCompression, ssEncrypted,
  ssIndexes, ssCachedUpdates, ssExtendedFieldProperties,
  ssBlobOnlyStreamEvent, ssAggregates, ssCustom);
```

Description
Allows for options to be set in saving in-memory datasets.

Option	Description
ssAstaLegacy	
ssZlibCompression	
ssEncrypted	
ssIndexes	
ssCachedUpdates	
ssExtendedFieldProperties	
ssBlobOnlyStreamEvent	
ssAggregates	
ssCustom	

1.8.1.1.2 Methods

TAstaDataSet

AddBookmarkIndex
AddIndex
AddIndexFields
ApplyRange
CancelRange
CleanCloneFromDataSet
CloneCursor
CloneFieldsFromDataSet
CloneFieldsFromDataSetPreserveFields
CompareFields
DataTransfer
DefineSortOrder
EditKey
EditRangeEnd
EditRangeStart
Empty
FastFieldDefine
FilterCount
FindKey
FindNearest
GetRecordSize
GotoKey
GotoNearest
IsBlobField
LastNamedSort
LoadFromFile
LoadFromFileWithFields
LoadFromStream
LoadFromStreamwithFields
LoadFromString
NukeAllFieldInfo
RemoveSortOrder
SaveToFile
SaveToStream
SaveToString
SetKey
SetRange
SetRangeEnd
SetRangeStart
SortOrderSort
UnRegisterClone
ValidBookmark

Derived from TDataSet
ActiveBuffer
Append
AppendRecord
BookmarkValid

Cancel
CheckBrowseMode
ClearFields
Close
CompareBookmarks
ControlsDisabled
CreateBlobStream
CursorPosChanged
Delete
DisableControls
Edit
EnableControls
FieldByName
FindField
FindFirst
FindLast
FindNext
FindPrior
First
FreeBookmark
GetBlobFieldData
GetBookmark
GetCurrentRecord
GetDetailDataSets
GetDetailLinkFields
GetFieldData
GetFieldList
GetFieldNames
GotoBookmark
Insert
InsertRecord
IsEmpty
IsLinkedTo
IsSequenced
Last
Locate
Lookup
MoveBy
Next
Open
Post
Prior
Refresh
Resync
SetFields
Translate
UpdateCursorPos
UpdateRecord
UpdateStatus

1.8.1.1.2.1 TAstaDataSet.AddBookmarkIndex

TAstaDataSet

Declaration
```
procedure AddBookmarkIndex;
```

Description
When an index is defined with no fields defined, the internal bookmarks will be indexed. This may increase performance with large DataSets using bookmarks.

1.8.1.1.2.2 TAstaDataSet.AddIndex

TAstaDataSet

Declaration
```
procedure AddIndex(FieldName: string; Descending: Boolean);
```

Description
Call AddIndex to create a new index for the client dataset based on a single field. FieldName is the field to use for the index. Descending specifies the sort order of the index.

1.8.1.1.2.3 TAstaDataSet.AddIndexFields

TAstaDataSet

Declaration
```
procedure AddIndexFields(TheIndexName: string; const FieldNames: array of
  string; const Descending: array of Boolean);
```

Description
Call AddIndexFields to create a new index for the client dataset.

TheIndexName is the name of the new index. FieldNames is an array of field names to include in the index. Descending is an array of Boolean values to define the sort order for each field in FieldNames. Descending must include the same number of values as FieldNames does.

1.8.1.1.2.4 TAstaDataSet.ApplyRange

TAstaDataSet

Declaration
```
procedure ApplyRange;
```

Description
Call ApplyRange to cause a range established with SetRangeStart and SetRangeEnd, or EditRangeStart and EditRangeEnd, to take effect. When a range is in effect, only those records that fall within the range are available to the application for viewing and editing.

After a call to ApplyRange, the cursor is left on the first record in the range.

1.8.1.1.2.5 TAstaDataSet.CancelRange

TAstaDataSet

Declaration
```
procedure CancelRange;
```

Description
Call CancelRange to remove a range currently applied to a client dataset. Canceling a range reenables access to all records in the dataset.

1.8.1.1.2.6 TAstaDataSet.CleanCloneFromDataSet

TAstaDataSet

Declaration
```
procedure CleanCloneFromDataSet(D: TDataSet); overload;
procedure CleanCloneFromDataSet(D: TDataSet; CallFirst: Boolean); overload;
```

Description
This method clears the current field definitions and copies the field definitions and data from the dataset D.

1.8.1.1.2.7 TAstaDataSet.CloneCursor

TAstaDataSet

Declaration
```
procedure CloneCursor(Source: TAstaCustomDataSet; KeepFilter: Boolean);
```

Description
TAstaDataSets may also be cloned. This is a process where the field definitions are copied to another TAstaDataSet and then linked to each other. Any edits, deletes or inserts will affect all clone linked datasets. The KeepFilter argument decides whether the Filter property and the OnFilterRecord event are used by the clone. You can remove all clone links by calling RemoveCloneLinks or just a specific one by calling UnRegisterClone.

See the CloneCursor Tutorial for an example of cloning.

1.8.1.1.2.8 TAstaDataSet.CloneFieldsFromDataSet

TAstaDataSet

Declaration
```
procedure CloneFieldsFromDataSet(SourceDataSet: TDataSet; AddData,
  IncludeBlobs: Boolean);
```

Description
This method clears the current field definitions and copies the field definitions from the SourceDataSet and allows Data to optionallaly be added as well as Blob Data.

1.8.1.1.2.9 TAstaDataSet.CloneFieldsFromDataSetPreserveFields

TAstaDataSet

Declaration

```
procedure CloneFieldsFromDataSetPreserveFields(D: TDataSet; AddData,
  IncludeBlobs, CallFirst: Boolean);
```

Description
Copies a dataset but does not clear out any previously defined fields.

1.8.1.1.2.10 TAstaDataSet.CompareFields

TAstaDataSet

Declaration
```
function CompareFields(key1, key2: TAstaDBlistItem; AField: TField;
  APartialComp, ACaseInsensitive: Boolean): Integer;
```

Description
Used internally for ASTA index support.

1.8.1.1.2.11 TAstaDataSet.DataTransfer

TAstaDataSet

Declaration
```
procedure DataTransfer(D: TDataSet; IncludeBlobs: Boolean);
```

Description
This method appends the current dataset with data from D. If blob data is required, then set IncludeBlobs to True.

1.8.1.1.2.12 TAstaDataSet.DefineSortOrder

TAstaDataSet

Declaration
```
procedure DefineSortOrder(ASortName: string; const AFields: array of
  string; const ADescending: array of Boolean);
```

Description
This method has been deprecated in Asta 3. Instead use the new method AddIndex.

1.8.1.1.2.13 TAstaDataSet.EditKey

TAstaDataSet

Declaration
```
procedure EditKey;
```

Description
Call EditKey to put the client dataset in dsSetKey state while preserving the current contents of the current search key buffer. To determine current search keys, you can use the IndexFields property to iterate over the fields used by the current index.

EditKey is especially useful when performing multiple searches where only one or two field values among many change between each search.

1.8.1.1.2.14 TAstaDataSet.EditRangeEnd

TAstaDataSet

Declaration
```
procedure EditRangeEnd;
```

Description
Call EditRangeEnd to change the ending value for an existing range. To specify an end range value, call FieldByName after calling EditRangeEnd. After assigning a new ending value, call ApplyRange to activate the modified range.

1.8.1.1.2.15 TAstaDataSet.EditRangeStart

TAstaDataSet

Declaration
```
procedure EditRangeStart;
```

Description
Call EditRangeStart to change the starting value for an existing range. To specify a start range value, call FieldByName after calling EditRangeStart. After assigning a new ending value, call ApplyRange to activate the modified range.

1.8.1.1.2.16 TAstaDataSet.Empty

TAstaDataSet

Declaration
```
procedure Empty;
```

Description
The Empty method deletes all records from the dataset. If the dataset is filtered, then only those rows that meet the filter condition as defined by the filter property or the OnFilterRecord event will be deleted.

1.8.1.1.2.17 TAstaDataSet.FastFieldDefine

TAstaDataSet

Declaration
```
procedure FastFieldDefine(FieldName: string; FType: TFieldType; Size:
  Integer);
```

Description
A fast way to add a new field to a TAstaClientDataSet. Just supply the field name, type and size and you new field will be created and added to the dataset.

1.8.1.1.2.18 TAstaDataSet.FilterCount

TAstaDataSet

Declaration
```
function FilterCount: Integer;
```

Description
FilterCount returns the count of all rows that meet the filter condition as defined by the Filter property and the OnFilterRecord event.

1.8.1.1.2.19 TAstaDataSet.FindKey

TAstaDataSet

Declaration
```
function FindKey(const KeyValues: array of const): Boolean;
```

Description
Call FindKey to search for a specific record in a dataset. KeyValues contains a comma-delimited array of field values, called a key. Each value in the key can be a literal, a variable, a NULL, or nil. If the number of values passed in KeyValues is less than the number of columns in the index used for the search, the missing values are assumed to be NULL

If a search is successful, FindKey positions the cursor on the matching record and returns True. Otherwise the cursor is not moved, and FindKey returns False.

1.8.1.1.2.20 TAstaDataSet.FindNearest

TAstaDataSet

Declaration
```
procedure FindNearest(const KeyValues: array of const);
```

Description
Call FindNearest to move the cursor to a specific record in a dataset or to the first record in the dataset that matches or is greater than the values specified in the KeyValues parameter. If there are no records that match or exceed the specified criteria, FindNearest positions the cursor on the last record in the table. The KeyExclusive property controls whether matching values are considered.

KeyValues contains a comma-delimited array of field values, called a key. If the number of values passed in KeyValues is less than the number of columns in the index used for the search, the missing values are assumed to be NULL.

Note: FindNearest works only with string data types.

1.8.1.1.2.21 TAstaDataSet.GetRecordSize

TAstaDataSet

Declaration
```
function GetRecordSize: Word;
```

Description
Returns the size of the current record.

1.8.1.1.2.22 TAstaDataSet.GotoKey

TAstaDataSet

Declaration

```
function GotoKey: Boolean;
```

Description
Use GotoKey to move to a record specified by key values assigned with previous calls to SetKey or EditKey and actual search values indicated in the search key buffer.

If GotoKey finds a matching record, it positions the cursor on the record and returns True. Otherwise the current cursor position remains unchanged, and GotoKey returns False.

1.8.1.1.2.23 TAstaDataSet.GoToNearest

TAstaDataSet

Declaration
```
procedure GotoNearest;
```

Description
Call GotoNearest to position the cursor on the record that is either the exact record specified by the current key values in the key buffer, or on the first record whose values exceed those specified. If there is no record that matches or exceeds the specified criteria, GotoNearest positions the cursor on the last record in the dataset.

Note: KeyExclusive determines which records are considered part of a search range.

Before calling GotoNearest, an application must specify key values by calling SetKey or EditKey to put the dataset is dsSetKey state, and then use FieldByName to populate the buffer with search values.

1.8.1.1.2.24 TAstaDataSet.IsBlobField

TAstaDataSet

Declaration
```
function IsBlobField(F: Tfield): Boolean;
```

Description
Call IsBlobField to determine whether a field represents BLOB data.

1.8.1.1.2.25 TAstaDataSet.LastNamedSort

TAstaDataSet

Declaration
```
function LastNamedSort: string;
```

Description
This method has been deprecated in Asta 3. See Indexes instead.

1.8.1.1.2.26 TAstaDataSet.LoadFromFile

TAstaDataSet

Declaration
```
procedure LoadFromFile(FileName: string);
```

Description
Loads a TAstaDataSet from a file. The fields should already have been defined using FieldsDefine. If the fields have NOT been defined then use LoadFromFileWithFields.

Related Topics
Streaming

1.8.1.1.2.27 TAstaDataSet.LoadFromFileWithFields

TAstaDataSet

Declaration
```
procedure LoadFromFileWithFields(const FileName: string);
```

Description
When AstaDataSets are loaded from a file, it is normally expected that the fields have already been defined, either from a select statement or using the FieldsDefine property. If fields have NOT been defined then use LoadFromFileWithFields.

Related Topics
Streaming

1.8.1.1.2.28 TAstaDataSet.LoadFromStream

TAstaDataSet

Declaration
```
procedure LoadFromStream(Stream: TStream);
```

Description
Loads a TAstaDataSet from a stream. Assumes that the fields have already been defined. If the fields have NOT been defined call LoadFromStreamwithFields.

Related Topics
Streaming

1.8.1.1.2.29 TAstaDataSet.LoadFromStreamwithFields

TAstaDataSet

Declaration
```
procedure LoadFromStreamWithFields(Stream: TStream);
```

Description
Although TAstaDataSets are saved with field information, when loading, the fields are normally thought as to have been already defined either from the FieldsDefine property or an SQL select statement. Use LoadFromStreamWithFields if you want the fields to be loaded from the stream.

Related Topics
Streaming

1.8.1.1.2.30 TAstaDataSet.LoadFromString

TAstaDataSet

Declaration
```
procedure LoadFromString(S: string);
```

Description
This loads just the data from a string into the dataset. The field definitions are not loaded with this method. The dataset should already contain matching field definitions.

Related Topics
Streaming

1.8.1.1.2.31 TAstaDataSet.NukeAllFieldInfo

TAstaDataSet

Declaration
```
procedure NukeAllFieldInfo;
```

Description
Removes all field definitions from a dataset.

1.8.1.1.2.32 TAstaDataSet.RemoveSortOrder

TAstaDataSet

Declaration
```
procedure RemoveSortOrder(SortName: string);
```

Description
This method has been deprecated in Asta 3. See Indexes instead.

1.8.1.1.2.33 TAstaDataSet.SaveToFile

TAstaDataSet

Declaration
```
procedure SaveToFile(const FileName: string);
```

Description
This saves a dataset to disk along with all field definitions and data.

Related Topics
Streaming

1.8.1.1.2.34 TAstaDataSet.SaveToStream

TAstaDataSet

Declaration
```
procedure SaveToStream(Stream: TStream);
```

Description
Saves a TAstaDataSet to a stream, including the field definitions.

Related Topics
Streaming

1.8.1.1.2.35 TAstaDataSet.SaveToString

TAstaDataSet

Declaration
```
function SaveToString: string;
```

Description
DataSets can be saved to a string and stored in blob fields.

1.8.1.1.2.36 TAstaDataSet.SaveToXML

TAstaDataSet

Declaration
```
procedure SaveToXML(const FileName: string; XMLFormat:
  TAstaXMLDataSetFormat); overload;
procedure SaveToXML(Stream: TStream; XMLFormat: TAstaXMLDataSetFormat);
  overload;

type TAstaXMLDataSetFormat = (taxADO, taxMidas);
```

Description
Use SaveToXML to save your dataset to either a file or stream in XML format. Specify an XMLFormat of taxADO for the XML format used by ADO, or taxMidas for the XML format used by MIDAS.

1.8.1.1.2.37 TAstaDataSet.SetKey

TAstaDataSet

Declaration
```
procedure SetKey;
```

Description
Call SetKey to put the dataset into dsSetKey state and clear the current contents of the search key buffer. Use FieldByName to supply the buffer then with a new set of values prior to conducting a search.

Note: To modify an existing key or range, call EditKey.

1.8.1.1.2.38 TAstaDataSet.SetRange

TAstaDataSet

Declaration
```
procedure SetRange(const StartValues, EndValues: array of const);
```

Description
Call SetRange to specify a range and apply it to the dataset. The new range replaces the currently specified range, if any.

StartValues indicates the field values that designate the first record in the range.

EndValues indicates the field values that designate the last record in the range.

SetRange combines the functionality of SetRangeStart, SetRangeEnd, and ApplyRange in a single procedure call. SetRange performs the following functions:

1. Puts the dataset into dsSetKey state.
2. Erases any previously specified starting range values and ending range values.
3. Sets the start and end range values.
4. Applies the range to the dataset.

After a call to SetRange, the cursor is left on the first record in the range.

If either StartValues or EndValues has fewer elements than the number of fields in the current index, then the remaining entries are set to NULL.

1.8.1.1.2.39 TAstaDataSet.SetRangeEnd

TAstaDataSet

Declaration
```
procedure SetRangeEnd;
```

Description
Call SetRangeEnd to put the dataset into dsSetKey state, erase any previous end range values, and set them to NULL. Subsequent field assignments made with FieldByName specify the actual set of ending values for a range.

After assigning end-range values, call ApplyRange to activate the modified range.

1.8.1.1.2.40 TAstaDataSet.SetRangeStart

TAstaDataSet

Declaration
```
procedure SetRangeStart;
```

Description
Call SetRangeStart to put the dataset into dsSetKey state, erase any previous start range values, and set them to NULL. Subsequent field assignments to FieldByName specify the actual set of starting values for a range.

After assigning start-range values, call ApplyRange to activate the modified range.

1.8.1.1.2.41 TAstaDataSet.SortDataSetByFieldName

TAstaDataSet

Declaration
```
procedure SortDataSetByFieldName(FieldName: string; Descending: Boolean);
```

Description
This method has been deprecated in Asta 3. Use Indexes now.

1.8.1.1.2.42 TAstaDataSet.SortDataSetByFieldNames

TAstaDataSet

Declaration
```
procedure SortDataSetByFieldNames(const AFieldNames: array of string; const
  ADescending: array of Boolean);
```

Description
This method has been deprecated in Asta 3. Use Indexes now.

1.8.1.1.2.43 TAstaDataSet.SortOrderSort

TAstaDataSet

Declaration
```
procedure SortOrderSort(SortName: string);
```

Description
This method has been deprecated in Asta 3. Use Indexes now.

1.8.1.1.2.44 TAstaDataSet.UnRegisterClone

TAstaDataSet

Declaration
```
procedure UnRegisterClone(Source: TAstaCustomDataSet);
```

Description
Call UnRegisterClone to remove a single cloned dataset's link to a source dataset. Clone linking is a process where the field definitions are copied to another Asta dataset and then linked to each other. Any edits, deletes or inserts will affect all clone linked datasets. To clone a dataset call CloneCursor.

1.8.1.1.2.45 TAstaDataSet.ValidBookMark

TAstaDataSet

Declaration
```
function ValidBookMark(BM: string): Boolean;
```

Description
Determines if a string is a valid dataset bookmark string.

1.8.1.1.3 Events

TAstaDataSet

OnStreamEvent

Derived from TDataSet
AfterCancel
AfterClose
AfterDelete
AfterEdit
AfterInsert
AfterOpen
AfterPost
AfterScroll
BeforeCancel
BeforeClose
BeforeDelete
BeforeEdit
BeforeInsert
BeforeOpen
BeforePost
BeforeScroll
OnCalcFields
OnDeleteError
OnEditError
OnFilterRecord
OnNewRecord
OnPostError

1.8.1.1.3.1 TAstaDataset.OnStreamEvent

TAstaDataSet

Declaration
```
property OnStreamEvent: TAstaStreamEvent;
```

Description
Allows for streams to be encrypted/compressed when SaveToStream or LoadToStream is called.

1.8.1.1.4 Functions for Working With TAstaDataSets

Unit
AstaDrv2

DataSetToString
StringToDataSet
CloneDataSetToString
DataSetToStringWithFieldProperties
FilteredDataSetToString
StringToDataSetWithFieldProperties

These additional helper functions allow you to manipulate the TAstaDataSet component in a variety of ways.

1.8.1.1.4.1 FilteredDataSetToString

Unit
AstaUtil

Declaration
```
function FilteredDataSetToString(D: TAstaDataSet): string;
```

Description
This allows a TAstaDataSet to be saved to a string and calls the TAstaDataSet method SaveToString but respects any active filters and only includes the rows matching the filter condition. To save the complete DataSet regardless of any active filters, use DataSetToString or CloneDataSetToString. The DataSet can be converted back to a TAstaDataSet by calling StringToDataSet.

1.8.1.1.4.2 DataSetToStringWithFieldProperties

Unit
AstaUtil

Declaration
```
function DataSetToStringWithFieldProperties(D: TAstaDataSet): string;
```

Description
This allows a TAstaDataSet to be saved to a string and also saves extended TField information.

1.8.1.1.4.3 StringToStream

Unit
AstaUtil

Declaration
```
procedure StringToStream(const S: string; var TM: TMemoryStream);
```

Description
Converts a String to a TMemoryStream that already exits.

1.8.1.1.4.4 StringToDataSetWithFieldProperties

Unit
AstaUtil

Declaration
```
function StringToDataSetWithFieldProperties(S: string): TAstaDataSet;
```

Description
This converts a string created using DataSetToStringWithFieldProperties to a TAstaDataSet and contains extended TField properties.

1.8.1.1.4.5 Sorting Datasets

These functions have been deprecated in ASTA 3 and replaced by Indexes

LastNamedSort
RemoveSortOrder
SortOrderSort
DefineSortOrder
SortDataSetByFieldName

1.8.1.1.4.6 DataSetToString

Unit
AstaUtil

Declaration
```
function DataSetToString(D: TAstaDataSet): string;
```

Description
This allows a TAstaDataSet to be saved to a string and calls the TAstaDataSet method SaveToString. The DataSet can be converted back to a TAstaDataSet by calling StringToDataSet.

1.8.1.1.4.7 CloneDataSetToString

Declaration
```
function CloneDataSetToString(D: TDataSet): string;
```

Description
This allows any TDataSet, not necessarily a TAstaDataSet descendent, to be copied to a string. It can then be converted into a TAstaDataSet by calling StringToDataSet.

1.8.1.1.4.8 StringToDataSet

Unit
AstaUtil

Declaration
```
function StringToDataSet(S: string): TAstaDataSet;
```

Description
This function calls the TAstaDataSet.LoadFromString method.

1.8.1.2 TAstaClientDataSet

Properties : Methods : Events

Unit
AstaClientDataset

Declaration
```
type TAstaClientDataSet = class(TAstaBaseClientDataSet)
```

Description
TAstaClientDataSet implements the client database functions on an ASTAClient. This is a hybrid component that implements methods and properties common to the two-tier TTable and TQuery components. It is the core "thin client" component if you are accessing a database. Multiple AstaClientDataSet objects connect to an AstaClientSocket that is connected to the ASTA server middle tier.
The TAstaClientDataSet behaves similar to a Delphi TQuery in that you normally enter some SQL and set Active to true to execute a query that fetches data from an Asta server. There is a Params property for parameterized queries and for use with StoredProcedures, MasterFields and MasterSource properties for no code Master/Detail support, a StoredProcedure property that auto populates the parameters, and the PrimeFields and TableName properties to allow for edit, insert and delete activitiy where ASTA generates the SQL. The TAstaClientDataSet can also be used while it is disconnected from an Asta server using the SuitcaseData property.

AstaClientDataSet objects connect to AstaClientSocket objects. See the Developer's Abstract.

1.8.1.2.1 Properties

TAstaClientDataSet

Aggregates
Ascending
AstaClientSocket
AutoFetchPackets
AutoIncrementField
DataBase
EditMode
ExtraParams
Indexes
IndexFieldCount
IndexFieldNames
IndexFields
IndexName
IndexPrimeKey
KeyExclusive
KeyFieldCount
MasterFields
MasterSource
MetaDataRequest
NoSQLFields
OldValuesDataSet
OracleSequence

OrderBy
ParamQueryCount
Params
PrimeFields
ProviderBroadCast
ProviderName
ReadOnly
RefetchOnInsert
ResetFieldsOnSQLChanges
RowsAffected
RowsToReturn
ServerDataSet
ServerMethod
ShowQueryProgress
SQL
SQLGenerateLocation
SQLOptions
SQLWorkBench
StoredProcedure
StreamOptions
SuitCaseData
TableName
UpdateMethod
UpdateMode
UpdateObject
UpdateTableName

Derived from TDataSet
Active
AggFields
AutoCalcFields
BlockReadSize
Bof
Bookmark
BufferCount
CanModify
Constraints
DataSetField
DataSource
DefaultFields
Designer
Eof
FieldCount
FieldDefList
FieldDefs
FieldList
Fields
FieldValues
Filter
Filtered
FilterOptions
Found

InternalCalcFields
IsUnidirectional
Modified
ObjectView
RecNo
RecordCount
RecordSize
SparseArrays
State

1.8.1.2.1.1 TAstaClientDataSet.Aggregates

TAstaClientDataSet

Declaration
```
property Aggregates: TAstaAggregates;
```

Description
Use Aggregates to define aggregates that summarize the data in the client dataset. Aggregates is a collection of TAstaIOAggregate objects, each of which defines a formula for calculating an aggregate value from a group of records in the client dataset. The individual aggregates can summarize all the records in the client dataset or subgroups of records that have the same value on a set of fields. Aggregates that summarize a subgroup of records are associated with indexes, and can only be used when the associated index is current.

TAstaIOAggregates is the type of the Aggregates property, which represents all the maintained aggregates for a client dataset. A single TAstaIOAggregate object represents each maintained aggregate. Maintained aggregates summarize data over the records in the client dataset.

TAstaIOAggregates can include a mix of active and inactive aggregate objects. Active aggregates are updated dynamically as the data in the client dataset is edited. Inactive aggregates define a formula for summarizing data and a group of records to summarize, but these formulas are not evaluated. Aggregates can be inactive because they are not currently needed, or because the index that defines the group of records they summarize is inactive.

1.8.1.2.1.2 TAstaClientDataSet.Ascending

TAstaClientDataSet

Declaration
```
property Ascending: Boolean
```

Description
The Ascending property is used in conjuction with the OrderBy property to allow you to write SQL with no order by statement and to easily change it at run time. See the OrderBy discussion for more detail.

1.8.1.2.1.3 TAstaClientDataSet.AstaClientSocket

TAstaClientDataSet

Declaration
property AstaClientSocket: TAstaClientSocket;

Description
An ASTA client application requires an AstaClientSocket. The AstaClientSocket "speaks" to the ASTA server as depicted in the Developer's Abstract.
In turn the AstaClientDataSet must communicate with the AstaClientSocket. The AstaClientSocket property is responsible for connecting the AstaClientDataSet to the AstaClientSocket. Note: Many AstaClientDataSets can be connected to a single AstaClientSocket.
This property should be set to the AstaClientSocket on your client. The property should fill in automatically when you drop the AstaClientDataSet onto your form or data module after you drop an AstaClientSocket component onto it. If you drop the AstaClientDataSet onto the form, and then the AstaClientSocket, you will have to set the property manually.

In the following diagram, the AstaClientDataSet1.AstaClientSocket property should be set to AstaClientSocket1.

1.8.1.2.1.4 TAstaClientDataSet.AutoFetchPackets

TAstaClientDataSet

Declaration
property AutoFetchPackets: Boolean

Description
When an ASTA server runs in a threaded state using the Persistent Sessions Threaded Model, a TAstaClientDataSet can be set up to open a cursor on the server and fetch rows of data on demand, rather than the complete dataset. Set the RowsToReturn property to a positive number and the SQLOptions.soPackets to True to enable this state.

When the above conditions are met, the AutoFetchPackets property will attempt to call GetNextPacket when a UI control is used, or there are any calls that reference EOF or the DoAfterScroll event. If AutoFetchPackets is set to False then you must manually call GetNextPacket.

1.8.1.2.1.5 TAstaClientDataSet.AutoIncrementField

TAstaClientDataSet

Declaration
```
property AutoIncrementField: string;
```

Description
The AutoIncrement property facilitates working with auto increment values. When you utilize ASTA's ability to automatically generate SQL, ASTA generates SQL for all the fields including your database's auto increment fields, but the auto increment field must not appear in Update or Insert statements, those values must be assigned by the database itself. Specifying the auto increment field in the AstaClientDataSet's AutoIncrementField property allows you to overcome this problem. When you specify the AutoIncrement field, ASTA does not include that field when it generates Update or Insert statements. The proper SQL is generated for the remaining fields and the database correctly assigns the auto increment value. If there are other fields that you do not want ASTA to include in Insert or Update statements, mark them as read only.

In order to allow you to work easily with auto increment values at the client (or to update those values if they are being displayed to the end user), ASTA provides an easy way for you to retrieve those values after an Insert statement. If the auto increment field is included as one of the fields in the RefetchOnInsert property, then the value in the AstaClientDataSet will be automatically updated after each transaction that includes any Insert statements.

If an AutoIncrementField is defined and included in this list of fields, then you must code the OnInsertAutoIncrementFetch event on the TAstaServerSocket. For Paradox and Access a call to Max (FieldName) is made. For SQLServer and SQL Anywhere you can code a call to the variable @@identity to retrieve the last auto increment value. The ASTA server then uses this value, or the prime key fields value to fetch all the fields as specified in the RefetchOnInsert property. The server SQL is executed and when it all is completed, the RefetchOnInsert values are streamed from the ASTA server and matched up to the appropriate rows in the TAstaClientDataSet.

Related Topics
ASTA SQL Generation
Oracle Sequence
SendMasterDetailAutoIncTransaction

1.8.1.2.1.6 TAstaClientDataSet.DataBase

TAstaClientDataSet

Declaration
```
property DataBase: string;
```

Description
This property applies to an AstaBDE server only. If you are using an AstaODBC server this property should be left blank.
If you are using the BDE, this property should be used if you wish to query different aliases from the same ASTA server. When using file server databases and the BDE, the DataBase property will map to the Alias property of a Delphi TQuery. The string value from the Database property is passed to the ASTA server so that you can query different

data sources from the server.

The following is an example of how the DataBase property appears to an ASTA server on a SubmitSQL statement coming from a TAstaClientDataSet:

```
procedure AstaServerSocket1SubmitSQL(Sender: TObject; DS: TDataSet;
  DataBaseStr, SQLString: string; Options: TAstaSelectSQLOptionSet);
```

1.8.1.2.1.7 TAstaClientDataSet.EditMode

TAstaClientDataSet

Declaration
```
property EditMode: string;
```

Description
By default, the AstaClientDataSets are read only. To enable editing of an AstaClientDataSet using client side SQL, you must set the EditMode property at design time or run time. At run time you can enable a dataset for editing with the SetEditMode method. When using TAstaProviders the EditMode property is not required as the SQLGenerateLocation when set to gsServer automatically puts the AstaClientDataSet into Cached EditMode.

At design time, click on the EditMode property's ellipse button to bring up the EditMode Property Editor. You must select the Update Method, the Prime Key/s (there can be more than one field in the prime key, which correlates to your database), and the UpdateTable (specify the table that you wish to update). The dataset's status will change to "Editable" when you have filled in all three of the required subproperties.

ASTA SQL Generation

1.8.1.2.1.8 TAstaClientDataSet.ExtraParams

TAstaClientDataSet

Declaration
```
function ExtraParams: TAstaParamList;
```

Description
These ExtraParams can be filled up and are used by AstaProviders when calling ApplyUpdates using AstaClientSocket. SendProviderTransactions and they then appear on AstaProviders on the server as AstaProvider.ClientDataSetExtraParams.

1.8.1.2.1.9 TAstaClientDataSet.Indexes

TAstaClientDataSet

Declaration
```
property Indexes: TAstaIndexes;
```

Description
Asta datasets can have indexes defined so that multiple sort orders can be defined. Indexes can be added at design time using the Indexes property or at runtime by calling AddIndex or AddIndexFields. The following properties and methods are used in conjunction with indexes:

Properties
IndexFieldCount
IndexFields
KeyExclusive
KeyFieldCount

Methods
ApplyRange
CancelRange
EditRangeEnd
EditRangeStart
SetRange
SetRangeEnd
SetRangeStart
SetKey
EditKey
FindKey
FindNearest
GotoKey
GotoNearest
AddIndexFields
AddIndex

1.8.1.2.1.10 TAstaClientDataSet.IndexFieldCount

TAstaClientDataSet

Declaration
```
property IndexFieldCount: Integer;
```

Description
Check IndexFieldCount to determine how many fields are used by the current index for the dataset.

1.8.1.2.1.11 TAstaClientDataSet.IndexFieldNames

TAstaClientDataSet

Declaration
```
property IndexFieldNames: string;
```

Description
Use IndexFieldNames as an alternative method of specifying the index to use for a client dataset. Specify the name of each field on which to index the dataset, separating names with semicolons. Ordering of field names is significant.

Indexes added using IndexFieldNames do not support grouping or maintained aggregates.

Note: The IndexFieldNames and IndexName properties are mutually exclusive. Setting one clears the other.

1.8.1.2.1.12 TAstaClientDataSet.IndexFields

TAstaClientDataSet

Declaration
```
property IndexFields[Index: Integer]: TField;
```

Description
IndexFields is a zero-based array of field objects, each of which corresponds to a field in the current index. Index is an ordinal value indicating the position of a field in the index. The first field in the index is IndexFields[0], the second is IndexFields[1], and so on.

Note: Do not set IndexField directly. Instead use the IndexFieldNames property to order datasets on the fly at runtime.

1.8.1.2.1.13 TAstaClientDataSet.IndexName

TAstaClientDataSet

Declaration
```
property IndexName: string;
```

Description
Use IndexName to specify an alternative index for a client dataset. If IndexName contains a valid index name, then that index is used to determine sort order of records.

Note: IndexFieldNames and IndexName are mutually exclusive. Setting one clears the other.

1.8.1.2.1.14 TAstaClientDataSet.IndexPrimeKey

TAstaClientDataSet

Declaration
```
property IndexPrimeKey: Boolean;
```

Description
Automatically adds an index for the primekey fields to aid in faster ProviderBroadcast updates and faster locates.

1.8.1.2.1.15 TAstaClientDataSet.KeyExclusive

TAstaClientDataSet

Declaration
```
property KeyExclusive: Boolean;
```

Description
Use KeyExclusive to specify whether a range includes or excludes the records that match its specified starting and ending values. By default, KeyExclusive is False, meaning that matching values are included.

To restrict a range to those records that are greater than the specified starting value and less than the specified ending value, set KeyExclusive to True.

1.8.1.2.1.16 TAstaClientDataSet.KeyFieldCount

TAstaClientDataSet

Declaration
```
property KeyFieldCount: Integer;
```

Description
Use KeyFieldCount to limit a search based on a multi-field key to a consecutive subset of the fields in that key. For example, if the primary key for a dataset consists of three-fields, a partial-key search can be conducted using only the first field in the key by setting KeyFieldCount to 1. Setting KeyFieldCount to 0 allows the client dataset to search on all key fields.

Note: Searches are only conducted based on consecutive key fields beginning with the first field in the key. For example if a key consists of three fields, an application can set KeyFieldCount to 1 to search on the first field, 2 to search on the first and second fields, or 3 to search on all fields. By default KeyFieldCount is initially set to include all fields in a search.

1.8.1.2.1.17 TAstaClientDataSet.MasterFields

TAstaClientDataSet

Declaration
```
property MasterFields: string;
```

Description
The MasterFields property is used for setting up master/detail relationships. This property must be used in conjunction with the MasterSource property and the detail table needs to use a parameterized query. A master/detail relationship can also be used in Suitcase mode using filters when the TAstaClientDataSet is disconnected from the server.

1.8.1.2.1.18 TAstaClientDataSet.MasterSource

TAstaClientDataSet

Declaration
```
property MasterSource: string;
```

Description
The MasterSource property determines the master table in a master/detail relationship. The property must be used in conjunction with the MasterFields property

1.8.1.2.1.19 TAstaClientDataSet.MetaDataRequest

TAstaClientDataSet

Declaration
```
property MetaDataRequest: TAstaMetaData;
```

Description
The MetaDataRequest property allows you to specify whether you are running a normal SQL query or are requesting specific meta data. See the ISQLDemo tutorial for examples of using the MetaDataRequest property.

Note: MetaData result sets are not supported at design time. Only at runtime.

Tip: You can get a result set which contains AstaServersocket.UserList information by setting the MetaDataRequest to mdUserList.

1.8.1.2.1.20 TAstaClientDataSet.NoSQLFields

TAstaClientDataSet

Declaration
```
property NoSQLFields: TStrings;
```

Description
Add any fields to NOSQLFields if you do not what those fields included in insert or update SQL generated by ASTA. Previous versions of ASTA required those fields to be tagged as read only.

See SQL Generation

1.8.1.2.1.21 TAstaClientDataSet.OldValuesDataSet

TAstaClientDataSet

Declaration
```
property OldValuesDataSet: TAstaDataSet;
```

Description
AstaClientDataSets maintain an internal FOldValuesDataSet which contains any deleted rows, newly inserted rows or the original values of any edited rows. This OldValuesDataSet allows support of CancelUpdates, RevertRecord and UpdatesPending.

Note: FOldValuesDataSet is the internal property that OldValuesDataSet references and is kept public for backward compatibility with old code.

1.8.1.2.1.22 TAstaClientDataSet.OracleSequence

TAstaClientDataSet

Declaration
```
property OracleSequence: string;
```

Description
Oracle sequences are different than auto increment fields in that they must be fired before an insert to retrieve a unique value. They can be set using the EditMode property of the TAstaClientDataSet from the Refetch tab or on the TAstaProvider if you are using server side SQL.
ASTA will use the OracleSequence string to call OracleSequence.NextValue before an insert statement. It will then retrieve it after the insert statement has successfully fired so that it can be fetched back to the client by calling OracleSequence.LastValue from the OnInsertAutoIncrementFetch event on the TAstaServerSocket.

ASTA supports the refetching of AutoIncrement Values on inserts. To support this the following steps must be taken:

1. Set the AutoIncrementField to the oracleSequenceField. With Oracle there is a problem at design time since ASTA is looking for Integer fields and Oracle does not have real integer fields but just Float fields that have 0 precision. So you must set this at runtime. (we'll look to change the property editor to adjust to this)
2. Set the Edit Mode to Cached. A Transaction is needed on the server so that this is mandatory.
3. Set the OracleSequence value in the EditMode property editor on the refetchTab or set it at runtime.
4. Click on the RefetchOnInsert Property and click on the field that will be used with the OracleSequence.

Here is an example of setting up a DataSet at runtime:

```
procedure TForm1.AstaClientDataSet1AfterOpen(DataSet: TDataSet);
begin
  with AstaClientDataSet1 do begin
    AutoIncrementfield := 'FG_ORDER_NO';
    OracleSequence := test_seq';
  end;
end;
```

You must then call ApplyUpdates(usmServerTransaction) to post the SQL to the server.

Here is what happens next.

1. the SQL is sent to the server and a Transaction is started.
2. for each Insert Statement the Insert is fired but the field Value for the AutoIncrement field will contain OracleSequence.NextVal.
3. After the Insert is fired, the AstaServerSocket.OnAutoIncrementFetch is fired and a result set is returned that has the current value from the oracle sequence just fired. Note: You must code the AutoIncrementFetch event on the server to call the correct sequence for the correct table!

```
procedure TfrmMain.AstaServerSocket1InsertAutoIncrementFetch(Sender:
TObject; ClientSocket: TCustomWinSocket; AQuery: TComponent;
TheDatabase, TheTable, TheAutoIncField: string; var AutoIncrementValue:
Integer);
begin
  with AQuery as TOracleDataSet do begin
    SQL.Clear;
    //right here you will have to do an if statement to get the right
sequence based on TheTable
    SQL.Add('select test_seq.CURRVAL from dual');
    Close;
    Open;
    if not Eof then AutoIncrementValue := Fields[0].AsInteger;
  end;
end;
```

4. Another select is called using the sequence to get any other values fired from triggers or default values that you had defined in the refetch on insert property on the client using the autoinc value just returned from the AutoIncrementFetch.

5. On the client, the results are streamed back and matched up by bookmarks. They can be retrieved immediately after their arrival in the AfterrefetchOnInsert of the

TAstaClientDataSet,

1.8.1.2.1.23 TAstaClientDataSet.OrderBy

TAstaClientDataSet

Declaration
```
property OrderBy: string;
```

Description
Use the OrderBy property to append an Order By clause onto your SQL statement. By keeping a separate order by clause you can easily change your SQL order by statement by changing this property, and then closing and opening the TAstaClientDataSet. By default, the OrderBy property is tagged with ASC for sort ascending, but if you have the Ascending property set to False, a DESC for descending will be applied.

1.8.1.2.1.24 TAstaClientDataSet.Params

TAstaClientDataSet

Declaration
```
property Params: TAstaParamList;
```

Description
The Params property holds parameter information and can be populated a number of different ways. AstaParamLists are fully streamable and are used extensively in the ASTA messaging system.
1. With paramaterized queries eg. Select * from Customer where custid = :CustomerNumber. :CustomerNumber will be extracted as a parameter
2. With stored procedures. When used this way the Params property will self populate as defined by the database.
3. With ExecSQLWithParams(S: string) to allow for insert statements that include binary or memo fields to be executed.
4. For use in general messaging using a TAstaClientSocket and TAstaServerSocket.
5. With ExecSQL.

Params are used in many different ways: with SQL that you provide, with stored procedures or to send values to Server Methods or Providers. They were designed to work just like the Delphi TQuery Params. The TAstaParamList is also an important part of the Asta Messaging feature as TAstaParamLists are fully streamable and can contain any kind of datatypes including datasets or other TAstaParamLists.

Here is an SQL example:

```
SELECT * FROM Customer WHERE CustNo = :CustomerNumber
```

The colon ":" is the token that the Params property keys off of to populate the parameter list.

Note: To allow ASTA to populate a parameterized query you must input the data type for each parameter. If you set the SQL to Select * from Customer, then set Active to True and then change the query to the above parameterized query, ASTA would be able to correctly run a query as it already had some field information. It knew that CustNo was a type ftInteger. If you initially write a parameterized query, click on the Params Property

Editor and set the data type to the correct value, whenever the SQL statement is changed, ASTA checks to see if any colons exist in the new SQL statement and internally calls Prepare to create the parameters. The default data type at this point for each parameter is ftString and the parameter type is set to ptInput.

1.8.1.2.1.25 TAstaClientDataSet.PrimeFields

TAstaClientDataSet

Declaration
```
property PrimeFields : TStrings;
```

Description
The PrimeFields property is used to provide information on the fields needed to fetch a row using a unique value. In order to have ASTA generate SQL automatically on edits, inserts and deletes, the PrimeFields must be defined. This is used by setting the EditMode property to define auto SQL generation and also by the PrimeFieldsWhereString method to return a Where statement for a given table that will apply to a unique row.

1.8.1.2.1.26 TAstaClientDataSet.ProviderBroadCast

TAstaClientDataSet

Declaration
```
property ProviderBroadCast: TProviderBroadCastOptions;
```

Description
The ProviderBroadCase property contains 3 sub properties RegisterForBroadcast used to register a client to receive broadcasts from a TAstaProvider, CachedBroadCastsWhenEdit, updates the OldValuesDataSet on broadcasts and BroadCastAction which allows you to control control what happens when a provider broadcast is received.

```
property BroadCastAction: TBroadCastAction;
property CacheBroadcastsWhenEdit: Boolean;
property MergeEditRowBroadCasts;
property ProviderFilter: string;
property RegisterForBroadCast: Boolean;
property RetrievePrimeKeyFields: Boolean;
```

See also Provider Broadcasts.

1.8.1.2.1.27 TAstaClientDataSet.ProviderBroadCast.BroadCastAction

TAstaClientDataSet. ProviderBroadcast

Declaration
```
property BroadCastAction: TBroadCastAction;

TBroadCastAction = (baIgnore, baCache, baAuto);
```

Description
BroadCastAction allows you to control how provider broadcast events are handled.

Value	Meaning
baIgnore	The OnReceiveProviderBroadCastEvent is not fired.
baAuto	If the OnReceiveProviderBroadCastEvent is coded then it is fired, otherwise if CacheBroadCastsWhenEdit is set to true and the state is not dsBrowse, then the OldValuesDataSet is updated rather than the actual dataset. Otherwise the DefaultApplyProviderBroadcast method is called and the DataSet is updated automatically.
baCache	The OldValuesDataSet is updated rather than the actual data.

1.8.1.2.1.28 TAstaClientDataSet.ProviderBroadCast.CachedBroadCastsWhenEdit

TAstaClientDataSet. ProviderBroadcast

Declaration
```
property CachedBroadCastsWhenEdit:Boolean;
```

Description
If the AstaClientDataSet.ProviderBroadCast. BroadCastAction is set to baAuto and CachedBroadCastsWhenEdit is set to true then any broadcasts received during an actually edit or insert operation will be cached by updating the OldValuesDataSet and not the current dataset that is being edited or inserted into.

1.8.1.2.1.29 TAstaClientDataSet.ProviderBroadCast.MergeEditRowBroadCasts

TAstaClientDataSet. ProviderBroadcast

Declaration
```
property MergeEditRowBroadcasts: Boolean;
```

Description
If this property is set to true, then provider broadcasts received for a row that is currently being edited will be automatically merged in. This merge will apply changes where the current editing has not already changed the value of that field. To override this functionality, define a ProviderBroadcastEditRow event handler. This property is only valid when CacheBroadcastsOnEdit is set to true, or BroadcastAction = baCache.

NOTE: If the user has changed a field but has not posted those changes (eg focus is still inside the TDBEdit control) then the value in the field they are currently editing will be updating to the value that was in the broadcast. If you want to take alternate action here, code the ProviderBroadcastEditRow event.

1.8.1.2.1.30 TAstaClientDataSet.ProviderBroadCast.ProviderFilter

TAstaClientDataSet. ProviderBroadcast

Declaration
```
property ProviderFilter: string;
```

Description
Broadcasts can be an expensive operation so by adding the ability to filter any datasets for broadcast can cut down on the amount of broadcasts. ProviderFilter:String supports normal TDataset filtering but is applied on the server. So if another user changes a dataset and updates a Cost Column, and a filter of Cost<10 is declared on the provider, and the cost =>10 there will be no broadcast.

1.8.1.2.1.31 TAstaClientDataSet.ProviderBroadCast.RegisterForBroadcast

TAstaClientDataSet. ProviderBroadcast

Declaration
```
property ProviderBroadcast.RegisterForBroadcast: Boolean;
```

Description
Setting this property registers a client to receive broadcasts from TAstaProviders.

1.8.1.2.1.32 TAstaClientDataSet.ProviderBroadCast.RetrievePrimeKeyFields

TAstaClientDataSet. ProviderBroadcast

Declaration
```
property RetrievePrimeKeyFields: Boolean;
```

Description
RetrievePrimekeyFields makes a call to SetPrimeKeyFields to retrieve primekey field info at runtime. This allows for the bsAuto option to be used to automatically update AstaClientDatasets receiving broadacasts.

In order to fetch primekey field information for any provider to allow the Provider.ProviderBroadcast.BroadcastAction to function as baAuto primekey field information must be available on the AstaClientDataSet. To retrieve primekey field information for providers, call the SetPrimeKeyFieldsFromProvider method after the Provider is opened and you are connected to the server. Use the OnAfterPopulate event to call this method.

```
if AstaClientDataSet1.PrimeFields.Count = 0 then
  AstaclientDataSet1.SetPrimeKeyFieldsFromProvider;
```

1.8.1.2.1.33 TAstaclientDataSet.ProviderName

TAstaClientDataSet

Declaration
```
property ProviderName: string;
```

Description

It is NOT required that you code ASTA servers to deploy an ASTA application but if you want to get more control over the SQL that ASTA generates or want to have access to update, insert and delete statements when they are fired on the server, then perhaps you need to use a TAstaProvider.
The ProviderName property shows a list of TAstaProviders that exist on an ASTA server. ASTA servers register their providers, server side datasets and TAstaBusinessObjectManagers by calling AstaServerSocket.RegisterDataModule.

1.8.1.2.1.34 TAstaClientDataSet.ReadOnly

TAstaClientDataSet

Declaration
```
property ReadOnly: Boolean;
```

Description
Use the ReadOnly property to prevent users from updating, inserting, or deleting data in the dataset. By default, ReadOnly is False, meaning users can potentially alter the dataset's data.

To guarantee that users cannot modify or add data to a dataset, set ReadOnly to True.

1.8.1.2.1.35 TAstaClientDataSet.RefetchOnInsert

TAstaClientDataSet

Declaration
```
property RefetchOnInsert: TStrings;
```

Description
The RefetchOnInsert property allows you to easily retrieve specific field values after an Insert or Update statement. If you have an auto increment field or perhaps a timestamp field that is inserted by a trigger, you can use this property to return those values. This property will work for a single transaction or a group of transactions.

Specify the fields that you want to retrieve in the RefetchOnInsert property editor. Whenever an Insert statement is fired, the values will be returned to the client. The AfterRefetchOnInsert event will notify you when the values have been returned. Code that event if you need to take immediate action with the returned value.

If an auto increment field is defined and included in this list of fields, then you must code the OnInsertAutoIncrementFetch event on the TAstaServerSocket. For Paradox and Access a call to Max (FieldName) is made. For SQLServer and SQL Anywhere you can code a call to the variable @@identity to retrieve the last autoincrement value. The ASTA server then uses this value, or the prime key fields values to fetch all of the fields as specified in the RefetchOnInsert property. All of the server SQL is executed and when it is completed, the RefetchOnInsert values are streamed from the ASTA server and matched up to the appropriate rows in the TAstaClientDataSet.

Note: This property will not work if your database does not support transactions.

Refetch Discussion

1.8.1.2.1.36 TAstaClientDataSet.ResetFieldsOnSQLChanges

TAstaClientDataSet

Declaration
```
property ResetFieldsOnSQLChanges: Boolean;
```

Description

1.8.1.2.1.37 TAstaClientDataSet.RowsAffected

TAstaClientDataSet

Declaration
```
property RowsAffected: Integer;
```

Description
ASTA 2.5 supports a public RowsAffected property on the AstaClientDataSet to be able to return the rows affected by ExecSQL statements.

See also TAstaServerSocket.OnExecRowsAffected.

1.8.1.2.1.38 TAstaClientDataSet.RowsToReturn

TAstaClientDataSet

Declaration
```
property RowsToReturn: Integer;
```

Description
The RowsToReturn property specifies how many rows to return in the result set. The default value is -1 which will return all rows. A value of 0 will return an empty DataSet. Other values will limit the result set to the RowsToReturn value. You may also set the number of rows that you want to populate at design time using the SQL WorkBench Design time Populate tab.
Note: this does not affect the server's query, only the amount packed up and dispatched by the ASTA server.

When an ASTA server runs in a threaded state using the Persistent Sessions Threaded Model, the RowsToReturn property allows you to open up a cursor on the server. SQL Options must have the soPackets set to True also. The initial fetch will return RowsToReturn rows, and then the dataset on the server will stay active allowing calls to GetNextPacket. You can close the Query on the server by calling CloseQueryOnServer.

1.8.1.2.1.39 TAstaClientDataSet.ServerDataSet

TAstaClientDataSet

Declaration
```
property ServerDataSet: string;
```

Description
ASTA supports the use of datasets on the ASTA server also. The property ServerDataSet pulls down a list of any TDataSet descendents in use on the connected ASTA server. If you are not sure as to which type any of these datasets are, you can get more detailed

information from the SQLWorkbench tab ServerDataSets. This shows not only the list of the TDataSets on the server but their Delphi 'type' also. A TQuery or a TTable or a TAstaDataSet are examples of possible values. This would include any TDataSet descendent that you use for your particular backend, like InfoPower TwwQueries or ODBCExpress TOEDataSets.
The ASTAClientDataSet can act as mirror of any of these datasets sitting on the ASTA server. Set the ServerDataSet property to the TDataSet descendent that you want to fetch data from and then set the MetaDataRequest property to TACustomDataSets. Note: design time population of ServerDataSets is not supported at this time. To populate the TAstaClientDataSet with the data contained in the specified server side TDataSet, just set the active property to True anytime after the TAstaClientSocket connects to the server. The earliest you may do this, is in the TAstaClientSocket's OnConnect event.

When using the ASTA server side features as described above there are additional options available that are not available when writing your own client side SQL.

The actual FieldDef properties from the ServerDataSet is streamed to the TAstaclientDataSet so that in the DoAfterOpen routine of the TAstaClientDataSet, all of the actual field properties, as set on the server are mirrored on the client. This includes such properties as DisplayLabel, DisplayWidth, Edit Masks, Alignment, ReadOnly, DisplayFormat, etc.There is slight additional overhead for this as all the FieldDef properties are streamed from the server to the client.

You have the opportunity to trigger a ServerActionEvent and to pass values to the ServerDataSet along with an Action Identifier. This will allow you to make decisions on the server according to client requests. To trigger a server side event for a specific dataset on the server, you must write two events, one on the client and one on the server.

The TAstaClientDataSet needs to have it's OnCustomServerActionEvent coded. The following example is from the tutorial found in the \Asta\Tutorials\ServerDataSet directory. The AstaClientDataSet in this example, is mirroring the UserDataSet on the ASTA server. The UserDataSet is an in-memory TAstaDataSet that contains a history of connect information that you can view from the Connected Users tab on the example ASTA servers. This event will force the UserDataSet to go to the first record and update the Activity field with the current date and time.

```
procedure TForm1.AstaClientDataSet1CustomServerAction(Sender: TObject;
  var ActionInt: Integer);
begin
  ActionInt := 17;
  //Params must be populated here. They will be set to 0 count before
  this call
  with AstaClientDataSet1 do begin
    Params.FastAdd('Asta Server Action!');
    Params.FastAdd(100);
    Params.FastAdd(2.35);
    Params.FastAdd(SysUtils.Now);
  end;
end;
```

The above example sends an ActionInt integer value to the server so that the server can trigger an event. By adding values to the Params property using the FastAdd method, typed parameters can be sent to the server. In this example, a String, an Integer, a

Double and a TDateTime are sent to the server.

The following code is taken from the BDEServer example code found in the \ASta\Servers\BDE\Code directory and is the actual code running on the BDE and ODBC servers as supplied with the ASTA components.

```
procedure TForm1.AstaServerSocket1ServerDataSetAction(Sender: TObject;
  DS: TDataSet; ActionID: Integer; ClientParams: TAstaParamList);
var
  i: Integer;
begin
  //this shows an example of a server side dataset that is being
  //streamed back to a client
  //You can check to see which dataset is being streamed and
  //take appropriate action based on the ActionID.
  //You also have incoming Params from the client if you want
  //to do something with them.
  if (DS = UserDataSet) and (Actionid = 17) and
    (UserDataSet.RecordCount>0) then
  begin
    UserDataSet.First;
    UserDataSet.Edit;
    UserDataSet.FieldByName('Activity').AsString :=
      'Client Action! at ' + DateTimeToStr(Now);
    UserDataSet.Post;
  end;
end;
```

The TAstaServerSocket has an OnServerDataSetAction event where you can compare the dataset coming in with the client request, process the ActionID coming in as an integer to key off of for any action you want to take, and also receive typed parameters coming from the client. You may want to close and open the dataset, change the Order By or Where clause or take any other kind of action. The parameters come in as a TAstaParamList so you may use them as any kind of data type that you need. In the above example, the incoming parameters are appended to the client requests memo on the server.

With this kind of server and client communication, ASTA gives you the best of client side AND server side development options. You decide how to write your 3-Tier application and ASTA supports any model that you want to use.

1.8.1.2.1.40 TAstaClientDataSet.ServerMethod

TAstaClientDataSet

Declaration
```
property ServerMethod: string;
```

Description
To create client applications that use no SQL with all of the business logic on the server, you can use a TAstaBusinessObjectsManager on any ASTA server. TAstaBusinessObjectManagers can define these server side "methods". AstaClientDataSets can choose which server side method to use, just by pulling down the ServerMethod property. When a server method is defined, paramaters can be defined also.

To invoke a server method at design time, pull down the ServerMethod property to see a list of methods available on the server. Any parameters defined on the server will be automatically created on the client. You can set these parameters at run time or at design time. When the Active property is set to True, the dataset will populate and any method "coded" on the server will be executed.

1.8.1.2.1.41 TAstaClientDataSet.ShowQueryProgress

TAstaClientDataSet

Declaration
```
property ShowQueryProgress: TShowQueryProgressType;
```

Description
The ShowQueryProgress property determines if a progress bar will appear when a TAstaClientDataSet is packaging up result set information.

Note: Use of this property does affect performance. The default value is sqDontShow.

1.8.1.2.1.42 TAstaClientDataSet.SQL

TAstaClientDataSet

Declaration
```
property SQL: TStrings;
```

Description
The SQL property allows you to enter SQL Queries. It operates just like the TQuery.SQL property. When used at design time, it will set the active property to False. To populate at design-time just set the Active property to True. If your SQL is valid, and you have an ASTAServer running, the AstaClientDataSet will populate. Once the dataset populates at design time ASTA will not attempt to refetch FieldDef information or data unless you change the SQL or you use the SQLWorkbench Property and click on the Design Populate button. This is to minimize unnecessary communication between to the server.

1.8.1.2.1.43 TAstaClientDataSet.SQLGenerateLocation

TAstaClientDataSet

Declaration
```
property SQLGenerateLocation: TGenerateSQLLocation;

type TGenerateSQLLocation = (gsClient, gsServer);
```

Description
An AstaClientDataSet can generate SQL on either the client or on the server. If you want complete control over the SQL generated by ASTA you can move the SQL generation to the ASTA server and get "between" each row of SQL that ASTA generates. To make an TAstaClientDataSet updatable that is connected to a TAstaProvider set the SQLGenerateLocation to gsServer and the EditMode will be set to Cached. The EditMode property editor is only used when using ClientSide SQL and not with TAstaProviders. TAstaProviders only support Cached Edit mode and to simulate AfterPost edit Mode behavior you can call ApplyUpdates from withing the AfterPost event.

1.8.1.2.1.44 TAstaClientDataSet.SQLOptions

TAstaClientDataSet

Declaration
```
property SQLOptions: TAstaSelectSQLOptionSet;
```

Description
When the soFetchMemos or soFetchBlobs subproperty is set to True, memo fields can be automatically fetched from a select statement. There are performance considerations when using this feature, particularly over a WAN connection, so please use it with caution. Any field that is a TMemo will be streamed from the server to the client. Please understand that overhead will be incurred when you use this feature. To activate the automatic fetching of memo fields, set the AstaDataDataSet.SQLOptions.soFetchMemos subproperty to True.

When EditMode is set to umAfterPost or umCached, and soFetchMemos and/or soFetchBlobs is set to true, then ASTA will automatically generate parameterized queries to insert or update to any blob or memo fields that are not tagged as readonly.

Tip: for performance purposes you may want to set soFetchMemos and soFetchBlobs to false when you do your select but then activate them after the AstaClientDataSet is opened so that any local edits and appends of blobs or memos are posted to the server in a parameterized query.

Example

```
AstaClientDataset.SQLOptions := AstaClientDataSet.SQLOptions +
  [soFetchMemos, soFetchBlobs];
```

1.8.1.2.1.45 TAstaClientDataSet.SQLWorkBench

TAstaClientDataSet

Declaration
```
property SQLWorkBench: string;
```

Description
The SQLWorkBench allows you to view the database's metadata while developing on the client. The SQLWorkBench allows you to discover critical information about your database.
This powerful property enables remote development; run an AstaServer at the remote location and an AstaClient can expose key database information - whether it's down the hallway, across town, or around the world.

Note: if you have added a field to a table on which you have right moused and defined persistent fields at design time use the SQL Workbench to force an update by clicking on the Design Populate button. Then Right Mouse from the AstaClientDataset at design time and use the Fields Editor to add the new field.

1.8.1.2.1.46 TAstaClientDataSet.StoredProcedure

TAstaClientDataSet

Declaration
```
property StoredProcedure: string;
```

Description
The StoredProcedure property allows you to select predefined StoredProcs. When a Stored Procedure is selected, the SQL and TableName properities will be cleared and the Params will automatically populate based on the database stored procedure definition.

AstaClientDataSets automatically create a parameters list when stored procedures are used. To use a stored procedure, make sure that the ASTA server that you are connected to is connected to a database that supports stored procedures. The DBDemos Paradox/Dbase alias that comes with Delphi does not support stored procedures.

The StoredProcedure property in an AstaClientDataSet is a pulldown property that queries the server metadata to retrieve a list of the stored procedures available. When you choose a stored procedure, the column names and additional information (datatype, input/output) is then fetched to enable ASTA to auto create a parameter list. This allows remote developers to use a stored procedure created on a database across the world, the moment after it was created and to transmit the parameter information to ASTA at design time!

When a stored procedure returns a result set when executed, the data will populate the TAstaClientDataSet. Any result parameters will be returned in the TAstaParamList. Stored procedures can be used at run time by calling the Prepare method after you change the StoredProcedure name so that ASTA can create the parameters for that stored procedure.

If your StoredProcedure does not return a result set then call ExecSQL. Calling ExecSQL with a the StoredProcedure property set to to a valid StoredProcedure name adds an soExecStoredProc to the SQLOptions of the AstaClientDataSet.SQLOptions property which is transmitted to the AstaServer so that the OnStoredProcedure event of the AstaServerSocket can be called with the NoResultSet:Boolean set to true. Setting the AstaClientDataSet.Active to true with the soExecStoredProc property set is equivalent to calling ExecSQL.

1.8.1.2.1.47 TAstaClientDataSet.StreamOptions

TAstaClientDataSet

Declaration
```
property StreamOptions: TAstaStreamSaveOptions;

TAstaStreamSaveOption = (ssAstaLegacy, ssZlibCompression, ssEncrypted,
  ssIndexes, ssCachedUpdates, ssExtendedFieldProperties,
  ssBlobOnlyStreamEvent, ssAggregates, ssCustom);
```

Description
AstaDataSets are streamable. By default AstaDatasets use the ASTA binary format to stream to disk and write all blobs. If ssIndexes is set to true then the Indexes are saved also. If ssAggregates is true then the Aggregates are saved. If ssBlobOnlyStream is true event an event is fired that passes any blob field so that it can be optionally compressed or uncompressed in the OnStreamEvent. Otherwise the OnStreamEvent includes the

complete DataSet stream including blobs and any indexes or aggregates.

ASTA 3 datasets are not stream compatible with ASTA 2 datasets as they write a header integer that includes the TAstaStreamSaveOptions. If ssExtendedFieldProperties are set to true than any persistent field properties are saved to the stream also.

ssZlibCompression, ssEncyrpted, and ssCachedUpdates are currently not implemented. The Asta StreamOptions tutorial shows how streams can be compressed optionally encrypted and then the appropriate flags can be set.

1.8.1.2.1.48 TAstaClientDataSet.SuitCaseData

TAstaClientDataSet

Declaration
```
property SuitCaseData: TSuitCaseData;
```

Description
The SuitCaseData property includes these three sub properites:
Active: Whether or not to use the suitcase model (Boolean).
FileName: The name of the file to which the dataset will be streamed to and from (String).
SavedCachedUpdates: Whether or not to save the OriginalValueDataSet that ASTA uses to implement cached updates (Boolean).

Active and FileName must be set in order for the suitcase model to be activated. Open will call the LoadFromFile(SuitCaseData.FileName) to load the dataset from disk. You must manually call the SaveToFile(SuitCaseData.FileName) to save the data to disk. If you want a TAstaClientDataSet to be open and NOT fetch data automatically from the server, set the SuitCaseData.Active property to True and FileName to NULL. OpenNoFetch will temporarily set the SuitCaseData.Active property to True to allow a TAstaClientDataSet to be opened without going to the server.

See also SuitCase Model and Streaming

1.8.1.2.1.49 TAstaClientDataSet.TableName

TAstaClientDataSet

Declaration
```
property TableName: string;
```

Description
The TableName property should be set to the table that you are targeting. This is not needed if you are writing an SQL query using the SQL property. Using the TableName property with the MetaDataRequest set to taNormalQuery creates an SQL statement of "SELECT * FROM [TableName]" (where TableName is your selection).

1.8.1.2.1.50 TAstaClientDataSet.UpdateMethod

TAstaClientDataSet

Declaration
```
property UpdateMethod: TAstaUpdateMethod;
```

```
type TAstaUpdateMethod = (umManual, umCached, umAfterPost);
```

Description
Determines when SQL is posted to the server. umAfterPost will fire after a record is posted. umCached requires the ApplyUpdates method to be called.

1.8.1.2.1.51 TAstaClientDataSet.UpdateMode

TAstaClientDataSet

Declaration
```
property UpdateMode: TAstaUpdateMode;

type TAstaUpdateMode = (upWhereAll, upWhereChanged, upWhereKeyOnly);
```

Description
This property specifies the way that ASTA generates the Where clause on update and delete statements when applying updates.

Value	Meaning
upWhereAll	Uses all the fields as they existed before the row was edited as found in the OldValuesDataSet.
upUpdateWhereKeyOnly	Uses just the primary key fields for the update statement.
upWhereChanged	Uses the primary key fields and the original value of fields that have changed for the where clause.

1.8.1.2.1.52 TAstaClientDataSet.UpdateObject

TAstaClientDataSet

Declaration
```
property UpdateObject: TAstaUpdateSQL;
```

Description
Use UpdateObject to specify the TAstaUpdateSQL component to use in an application that must be able to update a result set using the extended abilities of this component.

1.8.1.2.1.53 TAstaClientDateSet.UpdateTableName

TAstaClientDataSet

Declaration
```
property UpdateTableName: string;
```

Description
In order for ASTA to generate SQL on update, insert and delete statements, the UpdateTableName property must be set. To change the UpdateTableName of an AstaClientDataSet at run time use the SetEditMode method.

Related Topics

Asta SQL Generation

Multi Table Updates from Joins

1.8.1.2.2 Methods

TAstaClientDataSet

AddBookmarkIndex
AddParameterizedQuery
ApplyBulkUpdates
ApplyRange
AddIndex
AddIndexFields
CancelRange
CancelUpdates
CleanCloneFromDataSet
ClearParameterizedQuery
CloneCursor
CloneFieldsFromDataSet
CloneFieldsFromDataSetPreserveFields
CloseQueryOnServer
DataTransfer
DeleteNoCache
DeltaAsSQL
DeltaChanged
EditKey
EditRangeEnd
EditRangeStart
Empty
EmptyCache
ExecQueryInTransaction
ExecSQL
ExecSQLString
ExecSQLTransaction
ExecSQLWithInputParams
ExecSQLWithParams
FastFieldDefine
FetchBlob
FetchBlobString
FieldNameSendBlobToServer
FindNearest
FormatFieldforSQL
GetNextPacket
GetNextPacketLocate
GetRefetchStatus
GotoKey
GotoNearest
LoadFromFile
LoadFromFileWithFields
LoadFromStream
LoadFromStreamWithFields
LoadFromString

LoadFromXML
NukeAllFieldInfo
OpenNoFetch
OpenWithBlockingSocket
ParamByName
ParamQueryCount
PopSQLFromParser
Prepare
PrimeFieldsWhereString
PushSQLToParser
RefireSQL
RefireSQLBookmark
RefreshFromServer
RegisterProviderForUpdates
RemoveSortOrder
RevertRecord
SaveSuitCaseData
SaveToFile
SaveToStream
SaveToXML
SendBlobToServer
SendParameterizedQueries
SendSQLStringTransaction
SendSQLTransaction
SendStringAsBlobToServer
SendStringListToServer
SetEditMode
SetKey
SetPrimeKeyFieldsFromProvider
SetRange
SetRangeEnd
SetRangeStart
SetSQLString
SetToConnectedMasterDetail
SetToDisConnectedMasterDetail
SortDataSetByFieldName
SortDataSetByFieldNames
SortOrderSort
UnRegisterClone
UnRegisterProviderForUpdates
UpdatesPending

Derived from TDataSet
ActiveBuffer
Append
AppendRecord
BookmarkValid
Cancel
CheckBrowseMode
ClearFields
Close
CompareBookmarks
ControlsDisabled

CreateBlobStream
CursorPosChanged
Delete
DisableControls
Edit
EnableControls
FieldByName
FindField
FindFirst
FindLast
FindNext
FindPrior
First
FreeBookmark
GetBlobFieldData
GetBookmark
GetCurrentRecord
GetDetailDataSets
GetDetailLinkFields
GetFieldData
GetFieldList
GetFieldNames
GotoBookmark
Insert
InsertRecord
IsEmpty
IsLinkedTo
IsSequenced
Last
Locate
Lookup
MoveBy
Next
Open
Post
Prior
Refresh
Resync
SetFields
Translate
UpdateCursorPos
UpdateRecord
UpdateStatus

1.8.1.2.2.1 TAstaClientDataSet.AddBookmarkIndex

TAstaClientDataSet

Declaration
```
procedure AddBookmarkIndex;
```

Description
When an index is defined with no fields defined, the internal bookmarks will be indexed. This may increase performance with large DataSets using bookmarks.

1.8.1.2.2.2 TAstaClientDataSet.AddIndex

TAstaClientDataSet

Declaration
```
procedure AddIndex(FieldName: string; Descending: Boolean);
```

Description
Call AddIndex to create a new index for the client dataset based on a single field. FieldName is the field to use for the index. Descending specifies the sort order of the index.

1.8.1.2.2.3 TAstaClientDataSet.AddIndexFields

TAstaClientDataSet

Declaration
```
procedure AddIndexFields(TheIndexName: string; const FieldNames: array of
  string; const Descending: array of Boolean);
```

Description
Call AddIndexFields to create a new index for the client dataset.

TheIndexName is the name of the new index. FieldNames is an array of field names to include in the index. Descending is an array of Boolean values to define the sort order for each field in FieldNames. Descending must include the same number of values as FieldNames does.

1.8.1.2.2.4 TAstaClientDataSet.AddParameterizedQuery

TAstaClientDataSet

Declaration
```
procedure AddParameterizedQuery(TheSQL: string; P: TAstaParamList);
```

Description
When sending parameterized queries to be executed within on transaction on the server, AddParameterizedQuery adds a Query to the List to be executed.

1.8.1.2.2.5 TAstaClientDataSet.ApplyBulkUpdates

TAstaClientDataSet

Declaration
```
procedure ApplyBulkUpdates(RowsToApply: Integer)
```

Description
ApplyUpdates generates the SQL that has been generated for any edit, insert or delete activity since the last call to ApplyUpdates. If your server supports transactions, you may call ApplyUpdates(usmServerTransactions). If you have done a lot of updates and want to apply a subset of them you can use ApplyBulkUpdates. Note a separate transaction will be used for each call to this method.

1.8.1.2.2.6 TAstaClientDataSet.ApplyRange

TAstaClientDataSet

Declaration
```
procedure ApplyRange;
```

Description
Call ApplyRange to cause a range established with SetRangeStart and SetRangeEnd, or EditRangeStart and EditRangeEnd, to take effect. When a range is in effect, only those records that fall within the range are available to the application for viewing and editing.

After a call to ApplyRange, the cursor is left on the first record in the range.

1.8.1.2.2.7 TAstaClientDataSet.ApplyUpdates

TAstaClientDataSet

Declaration
```
function ApplyUpdates(TransactionMethod: TUpdateSQLMethod): string;
```

Description
ApplyUpdates generates the SQL that has been generated for any edit, insert or delete activity since the last call to ApplyUpdates. If your server support transactions, you may call ApplyUpdates(usmServerTransactions). Any error is returned as the function result but will also raise an exception. It is a function since it is used internally by other ASTA methods which require the error returned as a string from the server.

1.8.1.2.2.8 TAstaClientDataSet.CancelRange

TAstaClientDataSet

Declaration
```
procedure CancelRange;
```

Description
Call CancelRange to remove a range currently applied to a client dataset. Canceling a range reenables access to all records in the dataset.

1.8.1.2.2.9 TAstaClientDataSet.CancelUpdates

TAstaClientDataSet

Declaration
```
procedure CancelUpdates;
```

Description
When EditMode is set to anything other than Read Only; any edits, appends or deletes are recorded in a cache. Calling CancelUpdates rolls back any changes made, and puts the TAstaClientDataSet back in the same state it was in when it was first opened, or after ApplyUpdates was called last, and clears the cache. AstaClientDataSets use an internal OldValuesDataSet to implement this.

1.8.1.2.2.10 TAstaClientDataSet.CleanCloneFromDataSet

TAstaClientDataSet

Declaration
```
procedure CleanCloneFromDataSet(D: TDataSet);
```

Description
This method clears the current field definitions and copies the field definitions and data from the dataset D.

1.8.1.2.2.11 TAstaClientDataSet.ClearParameterizedQuery

TAstaClientDataSet

Declaration
```
procedure ClearParameterizedQuery;
```

Description
The AstaClientDataSet maintains an internal AstaParamList that contains the Parameterized Queries created with AddParameterizedQuery. ClearParameterizedQuery clears this internal list.

1.8.1.2.2.12 TAstaClientDataSet.CloneCursor

TAstaClientDataSet

Declaration
```
procedure CloneCursor(Source: TAstaCustomDataSet; KeepFilter: Boolean);
```

Description
TAstaClientDataSets may be cloned. This is a process where the field definitions are copied to another TAstaClientDataSet and then linked to each other. Any edits, deletes or inserts will affect all clone linked datasets. The KeepFilter argument decides whether the Filter property and the OnFilterRecord event are used by the clone. You can remove all clone links by calling RemoveCloneLinks or just a specific one by calling UnRegisterClone.

See the CloneCursor Tutorial for an example of cloning.

1.8.1.2.2.13 TAstaClientDataSet.CloneFieldsFromDataSet

TAstaClientDataSet

Declaration
```
procedure CloneFieldsFromDataSet(D: TDataSet; AddData, IncludeBlobs,
  CallFirst: Boolean);
```

Description
This method clears the current field definitions and copies the field definitions from the SourceDataSet and allows Data to optionallaly be added as well as Blob Data.

1.8.1.2.2.14 TAstaClientDataSet.CloneFieldsFromDataSetPreserveFields

TAstaClientDataSet

Declaration
```
procedure CloneFieldsFromDataSetPreserveFields(D: TDataSet; AddData,
  IncludeBlobs, CallFirst: Boolean);
```

Description
Copies a dataset but does not clear out any previously defined fields.

1.8.1.2.2.15 TAstaClientDataSet.CloseQueryOnServer

TAstaClientDataSet

Declaration
```
procedure CloseQueryOnServer;
```

Description
When fetching a result set with the AstaServer running in PersistentSessions mode, this will close the open cursor on the server.When an AstaClientDataset is destroyed CloseQueryOnServer is always called if all the rows have not been returned or it has not been previously called.

1.8.1.2.2.16 TAstaClientDataSet.CompareFields

TAstaClientDataSet

Declaration
```
function CompareFields(key1, key2: TAstaDBlistItem; AField: TField;
  APartialComp, ACaseInsensitive: Boolean): Integer;
```

Description
Used internally for ASTA index support.

1.8.1.2.2.17 TAstaClientDataSet.DataTransfer

TAstaClientDataSet

Declaration
```
procedure DataTransfer(D: TDataSet; IncludeBlobs: Boolean);
```

Description
This method appends the current dataset with data from D. If blob data is required, then set IncludeBlobs to True.

1.8.1.2.2.18 TAstaClientDataSet.DeleteNoCache

TAstaClientDataSet

Declaration
```
procedure DeleteNoCache;
```

Description
If EditMode is set to Cached this will delete a record and not post it to the server.

1.8.1.2.2.19 TAstaClientDataSet.DeltaAsSQL

TAstaClientDataSet

Declaration
```
function DeltaAsSQL: TStringList;
```

Description
DeltaAsSQL returns a TStringList that contains the SQL statements that ASTA generates for any edit, insert or delete activity. You are responsible for freeing the TStringList. This is used internally by ASTA's SQL generation and it is not something you would normally need to use.

Used Internally by ASTA. There is no real reason to call this directly.

1.8.1.2.2.20 TAstaClientDataSet.DeltaChanged

TAstaClientDataSet

Declaration
```
function DeltaChanged: Integer;
```

Description
Used internally by the TAstaClientDataSet and not normally something that needs to be used.
If you edit, insert or delete rows in a TAstaClientDataSet, ASTA keeps track of the status of any changes so that it knows whether or not to generate and post SQL to an ASTA server.
DeltaChanged returns an integer equivalent to the orginal values of the TDeltaType. If there is no change then DeltaChange returns -1.

```
type TDeltaType = (dtEdit, dtDelete, dtAppend);
```

1.8.1.2.2.21 TAstaClientDataSet.EditKey

TAstaClientDataSet

Declaration
```
procedure EditKey;
```

Description
Call EditKey to put the client dataset in dsSetKey state while preserving the current contents of the current search key buffer. To determine current search keys, you can use the IndexFields property to iterate over the fields used by the current index.

EditKey is especially useful when performing multiple searches where only one or two field values among many change between each search.

1.8.1.2.2.22 TAstaClientDataSet.EditRangeEnd

TAstaClientDataSet

Declaration
```
procedure EditRangeEnd;
```

Description
Call EditRangeEnd to change the ending value for an existing range. To specify an end range value, call FieldByName after calling EditRangeEnd. After assigning a new ending value, call ApplyRange to activate the modified range.

1.8.1.2.2.23 TAstaClientDataSet.EditRangeStart

TAstaClientDataSet

Declaration
```
procedure EditRangeStart;
```

Description
Call EditRangeStart to change the starting value for an existing range. To specify a start range value, call FieldByName after calling EditRangeStart. After assigning a new ending value, call ApplyRange to activate the modified range.

1.8.1.2.2.24 TAstaClientDataSet.Empty

TAstaClientDataSet

Declaration
```
procedure Empty;
```

Description
The Empty method deletes all records from the dataset. If the dataset is filtered, then only those rows that meet the filter condition as defined by the filter property or the OnFilterRecord event will be deleted.

1.8.1.2.2.25 TAstaClientDataSet.EmptyCache

TAstaClientDataSet

Declaration
```
procedure EmptyCache;
```

Description
If EditMode is in Cached this will clear the OldValuesDataSet.

1.8.1.2.2.26 TAstaClientDataSet.ExecQueryInTransaction

TAstaClientDataSet

Declaration
```
procedure ExecQueryInTransaction(pQuery: TAstaClientDataSet);
```

Description
Executes a parameterized query in a transaction using the passed in DataSet.

1.8.1.2.2.27 TAstaClientDataSet.ExecSQL

TAstaClientDataSet

Declaration
```
procedure ExecSQL;
```

Description
This is similar to the TQuery.ExecSQL for use with an SQL statement that does not return a result set. An exception will be raised if the ExecSQL was not successful. See ExecSQLWithParams and ExecSQLString if you do not want to change the SQL that you

have used in your select statement.

To execute stored procedures that do not return result sets you can call ExecSQL.

1.8.1.2.2.28 TAstaClientDataSet.ExecSQLString

TAstaClientDataSet

Declaration
```
procedure ExecSQLString(S: string);
```

Description
This is similar to the TQuery.ExecSQL for use with an SQL statement that does not return a result set. An exception will be raised if the ExecSQLString procedure was not successful. This allows you to issue some SQL to be executed on the server that doesn't use the Select SQL or the Params. To use Params and NOT use the Select SQL see ExecSQLWithParams. If you want to use the SQL in the SQL property use ExecSQL {linkN ==TAstaClientDataSet.ExecSQLString}

1.8.1.2.2.29 TAstaClientDataSet.ExecSQLTransaction

TAstaClientDataSet

Declaration
```
procedure ExecSQLTransaction;
```

Description
Same as ExecSQL but a transaction is started on the server with Commit called if successful and Rollback if not.

1.8.1.2.2.30 TAstaClientDataSet.ExecSQLWithInputParams

TAstaClientDataSet

Declaration
```
procedure ExecSQLWithInputParams(S: string; P: TAstaParamList);
```

Description
This allows the execution of ExecSQLwithParams with an AstaParamList that you must supply.

1.8.1.2.2.31 TAstaClientDataSet.ExecSQLWithParams

TAstaClientDataSet

Declaration
```
procedure ExecSQLWithtParams(S: string);
```

Description
This allows you to execute insert statements with binary and memo fields. Use this if you don't want to change the SQL of your select statement.

The following example is taken from the ExecSQLParam tutorial.

```
with NameDataSet do begin
```

```
    Params.Clear;
    //FastAdd adds a parameter and tries to set the data type according to
    the input
    //the first three come in as text, integer, datetime
    Params.FastAdd(Edit1.Text);
    Params.FastAdd(2);
    Params.FastAdd(Now);
    //the memo needs to be created explicitly
    Params.CreateParam(ftmemo, 'Description', ptInput);
    ParamByName('Description').AsMemo := Memo1.Text;
    Params.FastAdd(Time);
    //this shows how to send a paramerized update statement
    ExecSQLWithParams('Update Events Set Event_Name = :A, VenueNo = :B,
      Event_Date = :C, Event_Description = :D, Event_Time = :E Where
    EventNo = 1');
    Close;
    Open;
  end;
```

1.8.1.2.2.32 TAstaClientDataSet.FastFieldDefine

TAstaClientDataSet

Declaration

```
procedure FastFieldDefine(FieldName: string; FType: TFieldType; Size:
  Integer);
```

Description

A fast way to add a new field to a TAstaClientDataSet. Just supply the field name, type and size and you new field will be created and added to the dataset.

1.8.1.2.2.33 TAstaClientDataSet.FetchBlob

TAstaClientDataSet

Declaration

```
procedure FetchBlob(ATableName, FieldName, WhereString: string);
```

Description

When you call FetchBlob a TMemoryStream will be returned in the OnReceiveBlobStream event. Use FetchBlobString to return a blob as a string, directly forcing the TAstaClientDataSet to wait for the server to respond.

```
  procedure TForm1.AstaClientDataSet1ReceiveBlobStream(Sender: TObject;
    TableName, FieldName: string; var MS: TMemoryStream);
  begin
    //do something here with the MemoryStream. Asta disposes of the stream
    for you
  end;
```

1.8.1.2.2.34 TAstaClientDataSet.FetchBlobString

TAstaClientDataSet

Declaration

```
function FetchBlobString(ATableName, FieldName, WhereString: string):
  string;
```

Description
FetchBlobString returns a string directly from an ASTA server. You can use this for memo fields or even for binary fields like ftBlob by calling StringToStream or NewStringToStream from the AstaUtils unit.

1.8.1.2.2.35 TAstaClientDataSet.FieldNameSendBlobToServer

TAstaClientDataSet

Declaration
```
procedure FieldNameSendBlobToServer(FieldName: string; TMS: TMemoryStream);
```

Description
Sends a blob to an ASTAServer to be updated. PrimeFields and UpdateTablename must be defined in order for ASTA to be able to create the SQL Statement.

1.8.1.2.2.36 TAstaClientDataSet.FilterCount

TAstaClientDataSet

Declaration
```
function FilterCount: integer;
```

Description
FilterCount returns the count of all rows that meet the filter condition as defined by the Filter property and the OnFilterRecord event.

1.8.1.2.2.37 TAstaClientDataSet.FindKey

TAstaClientDataSet

Declaration
```
function FindKey(const KeyValues: array of const): Boolean;
```

Description
Call FindKey to search for a specific record in a dataset. KeyValues contains a comma-delimited array of field values, called a key. Each value in the key can be a literal, a variable, a NULL, or nil. If the number of values passed in KeyValues is less than the number of columns in the index used for the search, the missing values are assumed to be NULL

If a search is successful, FindKey positions the cursor on the matching record and returns True. Otherwise the cursor is not moved, and FindKey returns False.

1.8.1.2.2.38 TAstaClientDataSet.FindNearest

TAstaClientDataSet

Declaration
```
procedure FindNearest(const KeyValues: array of const);
```

Description
Call FindNearest to move the cursor to a specific record in a dataset or to the first record in the dataset that matches or is greater than the values specified in the KeyValues parameter. If there are no records that match or exceed the specified criteria,

FindNearest positions the cursor on the last record in the table. The KeyExclusive property controls whether matching values are considered.

KeyValues contains a comma-delimited array of field values, called a key. If the number of values passed in KeyValues is less than the number of columns in the index used for the search, the missing values are assumed to be NULL.

Note: FindNearest works only with string data types.

1.8.1.2.2.39 TAstaClientDataSet.FormatFieldforSQL

TAstaClientDataSet

Declaration
```
function FormatFieldforSQL(DS: TDataSet; FieldName: string): string;
```

Description
This returns a field formatted for an SQL statement according to the SQL settings in the TAstaClientSocket. Used internally by ASTA for SQL generation.

1.8.1.2.2.40 TAstaClientDataSet.GetNextPacket

TAstaClientDataSet

Declaration
```
function GetNextPacket: Integer;
```

Description
GetNextPacket attempts to return the next RowsToReturn number of rows, and will return a integer value representing the number of rows actually returned. If RowsToReturn is set to 50 and GetNextPacket returns any value less than 50, there are no more rows available on the server. When a TAstaClientDataSet is closed, or the TAstaClientSocket disconnects, all open server cursors are destroyed. If you want to explicity close a server cursor you may call CloseQueryOnServer. After this call, GetNextPacket will always return a zero value.

ASTA will internally attempt to fetch the next packets on demand even without an explicit call to GetNextPacket.

PacketFetches

1.8.1.2.2.41 TAstaClientDataSet.GetNextPacketLocate

TAstaClientDataSet

Declaration
```
function GetNextPacketLocate(const KeyFields: string; const KeyValues:
  Variant; Options: TLocateOptions): Integer;
```

Description
GetNextPacketLocate works like the TDataSet.Locate but it will use the query created from step #1. Depending on your database on the server, this can be a very fast operation. Using a file server database like DBISAM, locates will use indexes if the SQL does not include a join. The GetNextPacketLocate will call a TDataSet.Locate on the server and return RowsToReturn rows, and place the server side cursor on the position

where the locate has positioned it. This can result in very fast fetches on large tables. Your mileage will vary depending on how the database component used on the server handles this.

Packet Fetches

1.8.1.2.2.42 TAstaClientDataSet.GetRefetchStatus

TAstaClientDataSet

Declaration
```
function GetRefetchStatus: TRefetchStatus;
```

Description
Shows if the dataset is set to return field values after insert statements are executed on the server in a transaction.

1.8.1.2.2.43 TAstaClientDataSet.GotoKey

TAstaClientDataSet

Declaration
```
function GotoKey: Boolean;
```

Description
Use GotoKey to move to a record specified by key values assigned with previous calls to SetKey or EditKey and actual search values indicated in the search key buffer.

If GotoKey finds a matching record, it positions the cursor on the record and returns True. Otherwise the current cursor position remains unchanged, and GotoKey returns False.

1.8.1.2.2.44 TAstaClientDataSet.GoToNearest

TAstaClientDataSet

Declaration
```
procedure GotoNearest;
```

Description
Call GotoNearest to position the cursor on the record that is either the exact record specified by the current key values in the key buffer, or on the first record whose values exceed those specified. If there is no record that matches or exceeds the specified criteria, GotoNearest positions the cursor on the last record in the dataset.

Note: KeyExclusive determines which records are considered part of a search range.

Before calling GotoNearest, an application must specify key values by calling SetKey or EditKey to put the dataset is dsSetKey state, and then use FieldByName to populate the buffer with search values.

1.8.1.2.2.45 TAstaClientDataSet.LastNamedSort

TAstaClientDataSet

Declaration

```
function LastNamedSort: string;
```

Description
This method has been deprecated in Asta 3. See Indexes instead.

1.8.1.2.2.46 TAstaClientDataSet.LoadFromFile

TAstaClientDataSet

Declaration
```
procedure LoadFromFile(const FileName: string);
```

Description
Loads a TAstaClientDataSet from a file. The fields should already have been defined. If the fields have NOT been defined then use LoadFromFileWithFields.

1.8.1.2.2.47 TAstaClientDataSet.LoadFromFileWithFields

TAstaClientDataSet

Declaration
```
procedure LoadFromFileWithFields(const FileName: string);
```

Description
When TAstaClientDataSets are loaded from a file, it is normally expected that the fields have already been defined. If fields have NOT been defined then use LoadFromFileWithFields.

1.8.1.2.2.48 TAstaClientDataSet.LoadFromStream

TAstaClientDataSet

Declaration
```
procedure LoadFromStream(Stream: TStream);
```

Description
Loads a TAstaClientDataSet from a stream. Assumes that the fields have already been defined. If the fields have NOT been defined call LoadFromStreamwithFields.

1.8.1.2.2.49 TAstaClientDataSet.LoadFromStreamWithFields

TAstaClientDataSet

Declaration
```
procedure LoadFromStreamWithFields(Stream: TStream);
```

Description
Although TAstaClientDataSets are saved with field information, when loading, the fields are normally thought as to have been already defined with an SQL select statement. Use LoadFromStreamWithFields if you want the fields to be loaded from the stream.

1.8.1.2.2.50 TAstaClientDataSet.LoadFromString

TAstaClientDataSet

Declaration
```
procedure LoadFromString(S: string);
```

Description
This loads just the data from a string into the dataset. The field definitions are not loaded with this method. The dataset should already contain matching field definitions.

1.8.1.2.2.51 TAstaClientDataSet.LoadFromXML

TAstaClientDataSet

Declaration
```
procedure LoadFromXML(const FileName: string);
```

Description
Loads a dataset from an XML file supporting ADO or Midas formats.

1.8.1.2.2.52 TAstaClientDataSet.NukeAllFieldInfo

TAstaClientDataSet

Declaration
```
procedure NukeAllFieldInfo;
```

Description
Removes all field definitions from a dataset.

1.8.1.2.2.53 TAstaClientDataSet.OpenNoFetch

TAstaClientDataSet

Declaration
```
procedure OpenNoFetch;
```

Description
OpenNoFetch allows a TAstaClientDataSet to be opened empty without going to the server. To use this procedure you must have first opened the dataset at design time. This can be used for disconnected users allowing them to populate and edit a dataset, and then apply updates to a server at a later time.

1.8.1.2.2.54 TAstaClientDataSet.OpenWithBlockingSocket

TAstaClientDataSet

Declaration
```
procedure OpenWithBlockingSocket(TimeOut: Integer; RaiseException:
  Boolean);
```

Description
Since ASTA uses Non-blocking and event driven sockets which require windows messaging using ASTA Client components in an ISAPI dll has been problematic and we have supplied AstaWinManagement.pas in order to add a message pump to an ISAPI DLL but the results have not been consistent. So we have added some calls to allow the ASTA client socket to be used in blocking mode. To allow the ASTAClientDataSet to be used under ASTA 2.5 we have added routines to "cache" dataset calls if the AstaClientSocket.ClientType must be set to ctBlocking.

This means that once the AstaclientSocket is set to block you can call normal AstaclientDataSet calls and they will be cached until you say

TAstaClientSocket.SendAndBlock(100). Where 100 is a timeout value of how long to wait (see TWinSocketStream for details on this in the Delphi Help). Any error is returned as the function result.

If you want to raise an exception on the same call use TAstaClientSocket.SendAndBlockException and an exception will be raised.

The following are examples of how to use these new calls.

```
procedure TWebModule1.WebModule1WebActionItem1Action(Sender: TObject;
  Request: TWebRequest; Response: TWebResponse; var Handled: Boolean);
begin
  try
    AstaclientDataSet1.OpenWithBlockingsocket(500, True);
    Response.Content := '</HTML> RecordCount is ' +
  IntTostr(AstaClientDataSet1.RecordCount) + '  </HTML>';
  except
    Response.Content := '<HTML>' + Exception(ExceptObject).Message +
  '</HTML>';
  end;
  Handled := True;
end;

procedure TWebModule1.WebModule1WebActionItem2Action(Sender: TObject;
  Request: TWebRequest; Response: TWebResponse; var Handled: Boolean);
begin
  AstaClientSocket1.ClientType := ctBlocking;
  // you can now make as many calls as you want and they will be cached
  from the client
  AstaClientDataSet1.ExecSQLString('Update Customer set Company=' +
  QuotedStr(DateTimeToStr(now)) + ' where custno=1221');
  //they will all be sent when you send this
  try
    AstaClientSocket1.SendAndBlockException(100);
    Response.Content := '</HTML> Update Done </HTML>';
  except
    Response.Content := '<HTML>' + Exception(ExceptObject).Message +
  '</HTML>';
  end;
  Handled := True;
end;
```

1.8.1.2.2.55 TAstaClientDataSet.ParamByName

TAstaClientDataSet

Declaration
```
function ParamByName(ParamName: string): TAstaParamItem;
```

Description
This is equivalent to calling TAstaParamList.ParamByName.

1.8.1.2.2.56 TAstaClientDataSet.ParamQueryCount

TAstaClientDataSet

Declaration
```
function ParamQueryCount: Integer;
```

Description
Returns the count of the internal AstaParamList used after calling AddParameterizedQuery.

1.8.1.2.2.57 TAstaClientDataSet.PopSQLFromParser

TAstaClientDataSet

Declaration
```
procedure PopSQLFromParser;
```

Description
Uses the SQL previously sent to the AstaSQLParser.

1.8.1.2.2.58 TAstaClientDataSet.Prepare

TAstaClientDataSet

Declaration
```
procedure Prepare;
```

Description
Used at run time to create parameters on the fly from SQL or stored procedures. ASTA attempts to internally call prepare on SQL changes or a stored procedure name change.

1.8.1.2.2.59 TAstaClientDataSet.PrimeFieldsWhereString

TAstaClientDataSet

Declaration
```
function PrimeFieldsWhereString: string;
```

Description
This method will return a string that will fetch a unique row from a table.
If a customer table had a unique primary field on a CustNO: Integer column, the PrimeFieldsWhereString would return a string of 'WHERE Custno = 1000' if the current row had a CustNo value of 1000.

The primefields must be set first, usually using the EditMode property.

1.8.1.2.2.60 TAstaClientDataSet.PushSQLToParser

TAstaClientDataSet

Declaration
```
procedure PushSQLToParser;
```

Description
Takes the existing SQL and pushes it to an AstaSQLParser.

1.8.1.2.2.61 TAstaClientDataSet.ReadField

TAstaClientDataSet

Declaration
```
function ReadField(key: TAstaDBlistItem; AField: TField): Variant;
```

Description

1.8.1.2.2.62 TAstaClientDataSet.RefireSQL

TAstaClientDataSet

Declaration
```
procedure RefireSQL;
```

Description
RefireSQL calls DisableControls before it closes and opens the dataset so that there is no flicker to refetch data from an ASTA Server.

1.8.1.2.2.63 TAstaClientDataSet.ReFireSQLBookMark

TAstaClientDataSet

Declaration
```
procedure ReFireSQLBookMark;
```

Description
Calls RefireSQL and tries to go to the same bookmark which may or may not be valid.

1.8.1.2.2.64 TAstaClientDataSet.RefreshFromServer

TAstaClientDataSet

Declaration
```
procedure RefreshFromServer(SelectSQL: string); overload;
procedure RefreshFromServer(SelectSQL, WhereSQL: string); overload;
```

Description
If Primefields and UpdateTableName have been defined, RefreshFromServer will do a one row fetch to update the current row. Note: there is a performance cost to this as a Select statement is actually being performed on the server. ASTA will use the PrimeFields to buid the SQL where predicate but if you want to customize this even further you can use the overloaded RefreshFromServer(SelectSQL, WhereSQL: string).

1.8.1.2.2.65 TAstaClientDataSet.RegisterProviderForUpdates

TAstaClientDataSet

Declaration
```
procedure RegisterProviderForUpdates(Activate: Boolean);
```

Description
RegisterProviderForUpdates registers an AstaClientDataSet to be notified of any changes to a remote AstaProvider. By default, an AstaClientDataset will not be notified of any changes made by itself unless the AstaProvider.BroadCastsToOriginalClient property is

set to True.

RegisterProviderForUpdates is automatically called if the AstaClientDataSet.ProviderBroadCast.RegisterForBroadCast is set to True. To unregister an AstaClientDataSet call UnRegisterProviderForUpdates.

1.8.1.2.2.66 TAstaClientDataSet.RemoveSortOrder

TAstaClientDataSet

Declaration
```
procedure RemoveSortOrder(SortName: string);
```

Description
This method is now deprecated in Asta 3. Use Indexes instead.

1.8.1.2.2.67 TAstaClientDataSet.ResetFieldsOnSQLChanges

TAstaClientDataSet

Declaration
```
property ResetFieldsOnSQLChanges: Boolean;
```

Description
At design time or run time AstaClientDataSets reset their internal fields list on any change in SQL. If you are using persistent fields and know that the fields in any changed SQL will not change, set to this False. The default is True.

1.8.1.2.2.68 TAstaClientDataSet.RevertRecord

TAstaClientDataSet

Declaration
```
procedure RevertRecord;
```

Description
If you make any change to specific row, and have already posted it, you can call RevertRecord to undo those changes if the EditMode of the AstaClientDataSet has been set to umCached. AstaClientDataSets use an internal OldValuesDataSet to implement this.

1.8.1.2.2.69 TAstaClientDataSet.SaveSuitCaseData

TAstaClientDataSet

Declaration
```
procedure SaveSuitCaseData;
```

Description
This is equivalent to calling SaveToFile(SuitCaseData.FileName).

1.8.1.2.2.70 TAstaClientDataSet.SaveToFile

TAstaClientDataSet

Declaration

```
procedure SaveToFile(const FileName: string);
```

Description
This saves a dataset to disk along with all field definitions and data.

1.8.1.2.2.71 TAstaClientDataSet.SaveToStream

TAstaClientDataSet

Declaration
```
procedure SaveToStream(Stream: TStream);
```

Description
Saves a TAstaDataSet to a stream, including the field definitions.

1.8.1.2.2.72 TAstaClientDataSet.SaveToString

TAstaClientDataSet

Declaration
```
function SaveToString: string;
```

Description
DataSets can be saved to a string and stored in blob fields.

1.8.1.2.2.73 TAstaClientDataSet.SaveToXML

TAstaClientDataSet

Declaration
```
procedure SaveToXML(const FileName: string; XMLFormat:
  TAstaXMLDataSetFormat); overload;
procedure SaveToXML(Stream: TStream; XMLFormat: TAstaXMLDataSetFormat);
  overload;

type TAstaXMLDataSetFormat = (taxADO, taxMidas);
```

Description
Use SaveToXML to save your dataset to either a file or stream in XML format. Specify an XMLFormat of taxADO for the XML format used by ADO, or taxMidas for the XML format used by MIDAS.

1.8.1.2.2.74 TAstaClientDataSet.SendBlobToServer

TAstaClientDataSet

Declaration
```
procedure SendBlobToServer(ATableName, FieldName, WhereString: string;
  TMS: TMemoryStream);
```

Description
This sends a blob to an ASTA server and executes an update statement on the server. A more simpler approach would be to just create a TAstaClientDataSet and set the SQLOptions to soFetchBlobs and/or soFetchMemos, set the EditMode property and just use normal VCL methods to populate the dataset. To send a string to the server as a blob see SendBlobAsStringToServer.

1.8.1.2.2.75 TAstaClientDataSet.SendParameterizedQueries

TAstaClientDataSet

Declaration
```
function SendParameterizedQueies: TAstaParamList;
```

Description
Sends the list of Parameterized Queries created with AddParamaterizedQuery and exececutes it on an ASTA server in a transaction. The returning TAstaParamList will contain the SQL in the name of the Param and the RowsAffected AsInteger of the Param if the SQL is successful or the server Exception Message if the SQL was unsuccessful.

1.8.1.2.2.76 TAstaClientDataSet.SendSQLStringTransaction

TAstaClientDataSet

Declaration
```
function SendSQLStringTransaction(TransactionName: string; L: array of
  String): string;
```

Description
This sends a const array of Strings of SQL to the server and executes them in a transaction using the AstaServerSocket.OnTransactionStart call. An exception is raised if not successful and the string returned contains the exception message.

SendSQLTransaction

1.8.1.2.2.77 TAstaClientDataSet.SendSQLTransaction

TAstaClientDataSet

Declaration
```
function SendSQLTransaction(TransactionName: string; List: TStrings):
string;
```

Description
This sends a TStrings list of SQL to the server and executes them in a transaction using the AstaServerSocket.OnTransactionStart call. An exception is raised if not successful and the string returned contains the exception message.

1.8.1.2.2.78 TAstaClientDataSet.SendStringAsBlobToServer

TAstaClientDataSet

Declaration
```
procedure SendStringAsBlobToServer(ATableName, FieldName, WhereString, S:
string);
```

Description
This sends a blob to an ASTA server and executes an update statement on the server. A more simpler approach would be to just create a TAstaClientDataSet and set the SQLOptions to soFetchBlobs and/or soFetchMemos, set the EditMode property and just use normal VCL methods to populate the dataset. To send a stream to the server see SendBlobToServer.

1.8.1.2.2.79 TAstaClientDataSet.SendStringListToServer

TAstaClientDataSet

Declaration
```
procedure SendStringListToServer(ATableName, FieldName, WhereString:
string; S: TStrings);
```

Description
This sends a TStrings to an ASTA Server and executes an update statement on the server. A more simpler approach would be to just create a TAstaClientDataSet and set the SQLOptions to soFetchBlobs and/or soFetchMemos, set the EditMode property and just use normal VCL methods to populate the dataset. To send a stream to the server see SendBlobToServer. To send a string to the server as a blob see SendBlobAsStringToServer.

1.8.1.2.2.80 TAstaClientDataSet.SetEditMode

TAstaClientDataSet

Declaration
```
procedure SetEditMode(const Primes: array of string; TheUpdateTable:
  string; TheUpdateMethod: TAstaUpdateMethod);
```

Description
The SetEditMode method can be used to enable editing on an AstaClientDataSet at run time.

```
uses AstaDBTypes;

with AstaClientDataSet1 do begin
  Close;
  SQL.Clear;
  SQL.Add('SELECT * FROM EMPLOYEE');
  Open;
  SetEditMode(['EmpNo'], 'Employee', umAfterPost);
end;
```

Note: AstaClientDataSets can be enabled for edits at design time using the EditMode property.

1.8.1.2.2.81 TAstaClientDataSet.SetKey

TAstaClientDataSet

Declaration
```
procedure SetKey;
```

Description
Call SetKey to put the dataset into dsSetKey state and clear the current contents of the search key buffer. Use FieldByName to supply the buffer then with a new set of values prior to conducting a search.

Note: To modify an existing key or range, call EditKey.

1.8.1.2.2.82 TAstaClientDataSet.SetPrimeKeyFieldsFromProvider

TAstaClientDataSet

Declaration
```
procedure SetPrimeKeyFieldsFromProvider;
```

Description
Goes to the server and returns the primekey value for providers. Called internally if ProviderBroadcast.RetrievePrimekeyFields is set to true.

See Caching of Provider MetaData.

1.8.1.2.2.83 TAstaClientDataSet.SetRange

TAstaClientDataSet

Declaration
```
procedure SetRange(const StartValues, EndValues: array of const);
```

Description
Call SetRange to specify a range and apply it to the dataset. The new range replaces the currently specified range, if any.

StartValues indicates the field values that designate the first record in the range.

EndValues indicates the field values that designate the last record in the range.

SetRange combines the functionality of SetRangeStart, SetRangeEnd, and ApplyRange in a single procedure call. SetRange performs the following functions:

1. Puts the dataset into dsSetKey state.
2. Erases any previously specified starting range values and ending range values.
3. Sets the start and end range values.
4. Applies the range to the dataset.

After a call to SetRange, the cursor is left on the first record in the range.

If either StartValues or EndValues has fewer elements than the number of fields in the current index, then the remaining entries are set to NULL.

1.8.1.2.2.84 TAstaClientDataSet.SetRangeEnd

TAstaClientDataSet

Declaration
```
procedure SetRangeEnd;
```

Description
Call SetRangeEnd to put the dataset into dsSetKey state, erase any previous end range values, and set them to NULL. Subsequent field assignments made with FieldByName specify the actual set of ending values for a range.

After assigning end-range values, call ApplyRange to activate the modified range.

1.8.1.2.2.85 TAstaClientDataSet.SetRangeStart

TAstaClientDataSet

Declaration
```
procedure SetRangeStart;
```

Description
Call SetRangeStart to put the dataset into dsSetKey state, erase any previous start range values, and set them to NULL. Subsequent field assignments to FieldByName specify the actual set of starting values for a range.

After assigning start-range values, call ApplyRange to activate the modified range.

1.8.1.2.2.86 TAstaClientDataSet.SetSQLString

TAstaClientDataSet

Declaration
```
procedure SetSQLString(S: string);
```

Description
SetSQLString is a shortcut for setting the SQL property. It encapsulates the following two calls:

```
SQL.Clear;
SQL.Add(S);
```

1.8.1.2.2.87 TAstaClientDataSet.SetToConnectedMasterDetail

TAstaClientDataSet

Declaration
```
procedure SetToConnectedMasterDetail(SQLStr: string);
```

Description
This has the effect of setting up a detail parameterized query to allow a master/detail relationship to be used with the TAstaClientDataSet. Use this call at run time to toggle back and forth between a master/detail relationship that fetches the detail data from the remote server on each row change of the master table or when using filters on the detail table after fetching the complete detail dataset.

Example

```
with DetailDataSet do begin
  if Button.Tag = 0 then begin
    SetToDisconnectedMasterDetail('SELECT * FROM Orders ORDER BY
  OrderNo',
      'CustNo');
    Button.Tag := 1;
    Button.Caption : ='Set to run master/detail from Server';
  end
  else begin
    SetToConnectedMasterDetail('SELECT * FROM Orders WHERE Custno =
      :CustomerNumber');
    Button.Caption := 'Set to run master/detail Locally';
    Button.Tag := 0;
```

```
    end;
  end;
```

Master Detail Support
SetToDisconnectedMasterDetail

1.8.1.2.2.88 TAstaClientDataSet.SetToDisConnectedMasterDetail

TAstaClientDataSet

Declaration

```
procedure SetToDisconnectedMasterDetail(FullSelectString, DetailLinkField:
  string);
```

Description

This allows a master/detail relationship to be implemented by fetching the complete detail table and using local filters. Often performance can be improved using this technique.

Example

```
  with DetailDataSet do
    if Button.Tag = 0 then begin
      SetToDisconnectedMasterDetail('SELECT * FROM Orders ORDER BY ' +
        'OrderNo', 'CustNo');
      Button.Tag := 1;
      Button.Caption := 'Set to run master/detail from Server';
    end
    else begin
      SetToConnectedMasterDetail('SELECT * FROM ORDERS WHERE Custno ' +
        '= :CustomerNumber');
      Button.Caption := 'Set to run master/detail Locally';
      Button.Tag := 0;
    end;
```

Master Detail Support
SetToConnectedMasterDetail

1.8.1.2.2.89 TAstaClientDataSet.SortDataSetByFieldName

TAstaClientDataSet

Declaration

```
procedure SortDataSetByFieldName(FieldName: string; Descending: Boolean);
```

Description

Creates a temporary index on FieldName so that the DataSet appears in the order defined.

1.8.1.2.2.90 TAstaClientDataSet.SortDataSetByFieldNames

TAstaClientDataSet

Declaration
```
procedure SortDataSetByFieldNames(const AFieldNames: array of string; const
  ADescending: array of Boolean);
```

Description
Creates a temporary index so that the DataSet appears in the order defined.

1.8.1.2.2.91 TAstaClientDataSet.SortOrderSort

TAstaClientDataSet

Declaration
```
procedure SortOrderSort(SortName: string);
```

Description
This method has been deprecated in Asta 3. Use Indexes now.

1.8.1.2.2.92 TAstaClientDataSet.UnRegisterClone

TAstaClientDataSet

Declaration
```
procedure UnRegisterClone(Source: TAstaCustomDataSet);
```

Description
Call UnRegisterClone to remove a single cloned dataset's link to a source dataset. Clone linking is a process where the field definitions are copied to another Asta dataset and then linked to each other. Any edits, deletes or inserts will affect all clone linked datasets. To clone a dataset call CloneCursor.

1.8.1.2.2.93 TAstaClientDataSet.UnRegisterProviderForUpdates

TAstaClientDataSet

Declaration
```
procedure UnRegisterProviderForUpdates;
```

Description
Unregisters a provider for updates. See also RegisterProviderForUpdates.

1.8.1.2.2.94 TAstaClientDataSet.UpdatesPending

TAstaClientDataSet

Declaration
```
function UpdatesPending:Boolean;
```

Description
AstaClientDataSets maintain an internal OldValuesDataSet which contains any deleted rows, newly inserted rows or the original values of any edited rows. A call to

UpdatesPending, while EditMode is set to Cached, will check the OldValuesDataset and return true if any rows exist.

1.8.1.2.2.95 TAstaClientDataSet.ValidBookmark

TAstaClientDataSet

Declaration
```
function ValidBookMark(BM: string): Boolean;
```

Description
Determines if a string is a valid dataset bookmark string.

1.8.1.2.3 Events

TAstaClientDataSet

AfterRefetchOnInsert
OnAfterBroadcastHandling
OnAfterPopulate
OnBeforeBroadcastHandling
OnCommitAnySuccessError
OnCustomServerAction
OnProviderBroadcast
OnProviderBroadcastAfterApplyRow
OnProviderBroadcastAfterDeleteRow
OnProviderBroadCastEditRow
OnReceiveBlobStream
OnReceiveParams

Derived from TDataSet
AfterCancel
AfterClose
AfterDelete
AfterEdit
AfterInsert
AfterOpen
AfterPost
AfterScroll
BeforeCancel
BeforeClose
BeforeDelete
BeforeEdit
BeforeInsert
BeforeOpen
BeforePost
BeforeScroll
OnCalcFields
OnDeleteError
OnEditError
OnFilterRecord
OnNewRecord
OnPostError

1.8.1.2.3.1 TAstaClientDataSet.AfterRefetchOnInsert

TAstaClientDataSet

Declaration
```
property AfterRefetchOnInsert: TNotifyEvent;
```

Description
The OnRefetchOnInsert event fires after ASTA has done an insert on the server and any fields that were configured to be refetched are returned. For instance, if you have a master/detail set up and have the master table configured to refetch an auto increment field, you can fetch the auto increment field value as created on the server in the

AfterRefetchOnInsert event and then use this value to append the detail table.

1.8.1.2.3.2 TAstaClientDataSet.OnAfterBroadcastHandling

TAstaClientDataSet

Declaration
```
property OnAfterBroadcastHandling: TNotifyEvent;
```

Description
Called after any incoming Provider Broadcast.

When provider broadcasts are handled internally by ASTA the internal EditMode is set to umManual so that broadcast changes are not resubmitted to the server. The event above is called after editMode is set to umManual.

1.8.1.2.3.3 TAstaClientDataSet.OnAfterPopulate

TAstaClientDataSet

Declaration
```
property OnAfterPopulate: TNotifyEvent;
```

Description
The OnAfterPopulate event handler is called after the AstaClientDataSet has been populated.

1.8.1.2.3.4 TAstaClientDataSet.OnBeforeBroadcastHandling

TAstaClientDataSet

Declaration
```
property OnBeforeBroadcastHandling: TNotifyEvent;
```

Description
Called before any incoming provider broadcast.

When provider broadcasts are handled internally by ASTA the internal EditMode is set to umManual so that broadcast changes are not resubmitted to the server. The event above is called before editMode is set to umManual.

1.8.1.2.3.5 TAstaClientDataSet.OnCommitAnySucessError

TAstaClientDataSet

Declaration
```
property OnCommitAnySucessError: TAnyCommitAnySuccessEvent;
```

Description
Will return the error for any rows that raised an exception on the server when calling ApplyUpdates(usmCommitAnysuccess).

1.8.1.2.3.6 TAstaClientDataSet.OnCustomServerAction

TAstaClientDataSet

Declaration
```
property OnCustomServerAction: TCustomServerActionEvent
```

Description
This event is used to set up an action that can be triggered using ServerDataSets. The following example shows how an AstaParamList is populated and is set to trigger an "ActionID" integer to be sent to the ASTA server.

```
procedure TForm1.AstaClientDataSet1CustomServerAction(Sender: TObject;
  var ActionInt: Integer);
begin
  ActionInt := 17;
  //Params must be populated here.They will be set to 0 count before
  this call
  with AstaClientDataSet1 do begin
    Params.FastAdd('Asta Server Action!');
    Params.FastAdd(100);
    Params.FastAdd(2.35);
    Params.FastAdd(SysUtils.Now);
  end;
end;
```

1.8.1.2.3.7 TAstaClientDataSet.OnProviderBroadCast

TAstaClientDataSet

Declaration
```
type TProviderBroadCastEvent = procedure(Sender: TObject; D: TDataSet; var
DisposeOfDataSet: Boolean) of object; property OnProviderBroadCast:
TProviderBroadCastEvent
```

Description
When AstaClientDataSets are registered to receive broadcasts from providers, they will receive notification of any insert, delete or update activity from ASTA servers via a TAstaDataSet arriving in the OnProviderBroadCast event. This DataSet will have two additional fields. There will be a bookmark field that should be ignored as it will contain the bookmark of the original client dataset that made the change. It will also contain a Delta field that represents an integer as an ordinal of a TDeltaType
```
type TDeltaType = (dtEdit, dtDelete, dtAppend, dtAppendAndDelete);
```

Example
The incoming D: TDataSet is from the server and has the broadcast info. You can do anything you want to with the incoming dataset but ASTA will dispose of it if you don't set DisposeOfDataSet to False.

```
procedure TForm1.AstaClientDataSet1ProviderBroadCast(Sender: TObject;
  D: TDataSet; var DisposeOfDataSet: Boolean);
begin
  ChangeDS.DataSet.Free;
  ChangeDS.DataSet := D;
  DisposeOfDataSet := False;
end;
```

1.8.1.2.3.8 TAstaClientDataSet.OnProviderBroadCastAfterApplyRow

TAstaClientDataSet

Declaration
```
property OnProviderBroadCastAfterApplyRow: TProviderBroadCastNotifyEvent;
```

Description
Called after any row has been edited. Edit mode is umManual at this time.

1.8.1.2.3.9 TAstaClientDataSet.OnProviderBroadCastBeforeApplyRow

TAstaClientDataSet

Declaration
```
property OnProviderBroadCastBeforeApplyRow:
  TProviderBroadCastBeforeApplyRowEvent;
```

Description
Called before any row has been edited. Edit mode is umManual at this time.

1.8.1.2.3.10 TAstaClientDataSet.OnProviderBroadCastDeleteEditRow

TAstaClientDataSet

Declaration
```
property OnProviderBroadCastDeleteEditRow:
  TProviderBroadCastDeleteEditRowEvent;
```

Description
Called after any row has been deleted. Edit mode is umManual at this time.

1.8.1.2.3.11 TAstaClientDataSet.OnProviderBroadcastEditRow

TAstaClientDataSet

Declaration
```
property OnProviderBroadCastEditRow: TProviderBroadCastEditRowEvent

type TProviderBroadCastEvent = procedure(Sender: TObject; D: TDataSet; var
  DisposeOfDataSet: Boolean; var Handled: Boolean) of object;
```

Description
When MergeEditRowBroadcasts is set to true this event will be fired before the broadcast changes are applied to the current edit row.

The dataset D has fields:
"Field" - name of field changes
"OldValue" - value taken from OldValuesDataSet
"NewValue" - value taken from broadcast
"EditChanged" - true if the current row edit has already changed this field
"Apply" - set to true if you want ASTA to merge this change in automatically.

NOTE: If the user has changed a field but has not posted those changes (eg focus is still inside the TDBEdit control) then EditChanged will be false even though the user may have changed the field. If you want to cater for this case you need to code this event and perhaps take action based on TForm.ActiveControl.

Each change is listed in a separate row. If you handle the merge yourself, set the Handled parameter to true and ASTA will not do any merging itself.
If you wish to keep the dataset, set the DisposeOfDataSet parameter to false.

1.8.1.2.3.12 TAstaClientDataSet.OnReceiveBlobStream

TAstaClientDataSet

Declaration
```
property OnReceiveBlobStream: TReceiveBlobEvent;
```

Description
The OnRevceiveBlobStream provides the mechanism to recieve a TMemoryStream when FetchBlob is called. The more accepted method of fetching a blob is to use FetchBlobString.

1.8.1.2.3.13 TAstaClientDataSet.OnReceiveParams

TAstaClientDataSet

Declaration
```
property OnReceiveParams: TNotifyEvent;
```

Description
When using StoredProcedures or ServerMethods an Params defined as ptOutput,ptInputOutPut and ptResult ParamTypes come back from ASTA servers. They will arrive in the OnReceiveParams Event.

1.8.1.2.3.14 TAstaClientDataSet.OnStreamEvent

TAstaClientDataSet

Declaration
```
property OnStreamEvent: TAstaStreamEvent;
```

Description
Allows for streams to be encrypted/compressed or when SaveToStream and LoadToStream is called.

1.8.1.3 TAstaClientSocket

Properties : Methods : Events

Unit
AstaClientSocket

Declaration
```
type TAstaClientSocket = class(TAstaCustomClientSocket);
```

Description
The ASTA client to the ASTA server. All communcation between client and server is facilitated by the AstaClientSocket. ASTA supports a sophisticated messaging layer as well as database calls. In Database terms think of the TAstaClientSocket as a Database Connection to the Server. AstaClientSocket's are async and event driven so there is no threading.

The Address and Port properties on the ASTA client must match the Address and Port properties on the ASTA server.

The AstaClientSocket is the component that connects the client to the AstaServer or middle-tier. It descends from a combination of the VCL's socket objects. The AstaClientSocket is the client end of the "pipe". All data sent to or from the application passes through this component. The AstaClientSocket's Address and Port properties need to match the server's Address and Port properties (the Host and Port properties can be used if you are using the Domain Naming System [DNS]).

The AstaClientSocket, is also a place where "global" characteristics can be assigned to the client piece of the application. If you wish to use compression or encryption, for instance, you would set those at the AstaClientSocket.

See also: Sockets and TCP/IP Issues

See the Developer's Abstract to see how the AstaClientSocket fits into the ASTA architecture.

1.8.1.3.1 Properties

TAstaClientSocket

Active
Address
AnchorStatus
ApplicationName
ApplicationVersion
AstaServerVersion
AutoLoginDlg
ClientSocketParams
ClientType
Compression
ConnectAction
Connected

ConnectionString
CursorOnQueries
DataSetCount
DataSets
DTPassword
DTUserName
Encryption
Host
Password
Port
ProgressBar
SocksServerAddress
SocksServerPort
SocksUserName
SQLDialect
SQLErrorHandling
SQLOptions
SQLTransactionEnd
SQLTransactionStart
StatusBar
UpdateSQLSyntax
UserName

1.8.1.3.1.1 TAstaClientSocket.Active

TAstaClientSocket

Declaration
```
property Active: Boolean;
```

Description
When the Active property is set to True, a connection to the Asta server is established. In order for the connection to complete properly, the Address (or Host) and the Port properties must point to a running Asta server.
If the ASTA server is running at IP address 208.234.120.25 and at port 9000, then the AstaClientSocket Address property should be set to 208.230.120.25 and the Port property should be set to 9000.

Since the AstaClientSocket is an asyncronous non-blocking socket, setting it to Active does not necessary insure that it actually is Active as the time it takes to actually connect to the server and become active will vary according to network considerations. You know exactly when you are connected to the server when the OnConnect Event fires or you call OpenTheSocket.

Early Connect

1.8.1.3.1.2 TAstaClientSocket.Address

TAstaClientSocket

Declaration
```
property Address: string;
```

Description
The Address property must be set to the IP address of the Asta server. For instance, if you are developing on a single machine, you would set the address to 127.0.0.1 (the local loopback address). If your server is on a second system then you should use the IP Address of that system.
If you are using DNS (the Domain Naming System), and can address your systems by name ("MYSERVER.MYCOMPANY.COM"), then you should use the Host property and leave the Address property blank. If you set the Address and the Host, the Host property will take precedence.

IMPORTANT: This property must be used in conjunction with the Port property.

1.8.1.3.1.3 TAstaClientSocket.AnchorStatus

TAstaClientSocket

Declaration
```
property AnchorStatus: TClientAnchorStatus;

type TClientAnchorStatus = (TACNoAnchor, TACConnectedToAnchor,
  TACConnectedToServer, TACDisconnectedFromAnchor,
  TACDisconnectedFromServer, TAAnchorReDirect);
```

Description
Shows the current state of an AstaClientSocket if it uses an Asta Anchor Server.

1.8.1.3.1.4 TAstaClientSocket.ApplicationName

TAstaClientSocket

Declaration
```
property ApplicationName :string;
```

Description
Allows you to specify an application name to send to the server. You might wish to use this property if your ASTA server is serving several different client applications at the same time.
This property is also used by ASTA's Automatic Client Update feature. It is used in conjunction with the ApplicationVersion property.

1.8.1.3.1.5 TAstaClientSocket.ApplicationVersion

TAstaClientSocket

Declaration
```
property ApplicationVersion: string;
```

Description
This ApplicationVersion is used with the ApplicationName property to enable ASTA's Automatic Client Update feature.
Starting with ASTA v 1.91, if you have distributed 50 copies of an Asta client as version 1.0, you can automatically update them by registering a later Asta client version, say 1.1 or 2.0, at the server. When a user of version 1.0 logins into the server, he or she will be automatically updated to the latest version of your software.
To enable an automatic update, provide a version number like "1.0" in your initial release. When you are ready for the next release, increment the ApplicationVersion property to "1.1" or "2.0" and recompile your program. The resulting executable file should then be registered at the server. After you have registered the program, any 1.0 client that logs in will be automatically updated to the version registered at the server. It is critically important that you remember to increment the ApplicationVersion or no update will occur.

NOTE: The ApplicationVersion property is a STRING. When it is compared at the server it is converted to a float. 1.1 will be greater than 1.0. 1.1111 will be greater than 1.1. Use 1.0, 1.1, 1.2 and do not use 1.1.1.1 or 1.1.1.2, etc.

1.8.1.3.1.6 TAstaClientSocket.AstaServerVersion

TAstaClientSocket

Declaration
```
property AstaServerVersion: Double;
```

Description
When an AstaClientSocket connects to an ASTA server, AstaServerVersion is returned as defined in AstaAbout as Asta version

1.8.1.3.1.7 TAstaClientSocket.AutoLoginDlg

TAstaClientSocket

Declaration
```
property AutoLoginDlg: TLoginType;
```

Description
AutoLoginDlg is the foundation for automating client logins. There are three options:

Value	Meaning
ltLoginDlg	Displays a user name and password dialog, must be verified at the server.
ltLoginNoDlg	Passes UserName and Password properties, must be verified at the server.
ltNoChallenge	UserName and Password properties are transmitted, automatically verified at server.

ASTA clients by default do not Login to the server. You must set the AutoLoginDlg to secure your ASTA server. Note: A Login is required to trigger Automatic Client Updates.

Use ltLoginDlg or ltLoginNoDlg for projects that need verification. ltLoginDlg raises a dialog that prompts for the clients user name and password. The inputs are assigned to the UserName and Password properties. The values can be checked at the server in the OnClientAuthenticate event. If the value is acceptable, set the verified parameter to True. ltLoginNoDlg sends the values assigned the UserName and Password properties and passes the information to the server. If the user is to be logged into the system, the OnClientAuthenticate event must set the verified parameter to True.
Use ltNoChallenge when you simply don't want to screen the users at the server. No verification is required in the OnClientAuthenticate event. The user is automatically accepted at the server.

1.8.1.3.1.8 TAstaclientSocket.ClientSocketParams

TAstaClientSocket

Declaration
```
property ClientSocketParams: TAstaParamList;
```

Description
The ClientSocketParams is a "scratch" TAstaParamList that you can fill with any kind of information that will then be transported to the AstaServerSocket and added to the AstaServerSocket.UserList.GetUserParamList. By using the ClientSocketParams you can have any kind of additional information available on the AstaServer.

1.8.1.3.1.9 TAstaClientSocket.ClientType

TAstaClientSocket

Declaration
```
property ClientType: TClientType;
type TClientType = (ctNonBlocking, ctBlocking);
```

Description
In ASTA applications, this property should be set to "ctNonBlocking". See the Delphi help under TClientSocket for background.

1.8.1.3.1.10 TAstaClientSocket.Compression

TAstaClientSocket

Declaration
```
property Compression: TAstaCompression;
```

Description
This property determines if compression will be employed by your application. It works in conjunction with the OnCompress and OnDecompress events.

NOTE: This property must have the SAME SETTING on the ServerSocket AND the ClientSocket.

1.8.1.3.1.11 TAstaClientSocket.ConnectAction

TAstaClientSocket

Declaration
```
property ConnectAction: TAstaConnectAction;
```

Description
Specify the dialog, if any, that the user will see when starting the program and trying to connect to the ASTA server.

Value	Meaning
caFastConnectCombo	Select from several servers, allows new server entry.
caFastConnect	Last IP used is the one displayed. New IPs can be entered.
caCustomConnect	Allows for a custom login dialog. See the note at bottom.
caUseDesignAddress	No dialog is displayed, uses the IP entered at design time.
caNone	No dialog is displayed, developer is responsible for assigning it.

Many developers want to create a custom login screen for their users. A custom login provides you with the opportunity to present a promotional splash screen or even an advertising opportunity. If you select the caCustomConnect option of the AstaClientSocket's ConnectAction property, then you can customize a login screen. If you select this option, you can display your login screen from within the OnCustomConnect event handler. In that event handler, you have access to the AstaClientSocket and all of its properties. You must set the Address and Port properties or the Host and Port properties. Those properties become your responsibility when implementing a custom login.procedure.

```
TForm1.AstaClientSocket1CustomConnect(Sender: TObject);
begin
  //CALL YOUR CUSTOM LOGIN SCREEN HERE
  //SET USERNAME, PASSWORD, AND OTHER PROPERTIES
  // YOU MUST SET THE ADDRESS (OR HOST) AND PORT
```

```
    AstaClientSocket1.Address := '206.210.19.131';
    AstaClientSocket1.Port := 8080;
  end;
```

1.8.1.3.1.12 TAstaClientSocket.Connected

TAstaClientSocket

Declaration
```
function Connected: Boolean;
```

Description
Checks the AnchorStatus property and responds True if equal to tacConnectedToServer. This should only be used when the AstaAnchorServer is running.

Use the Asta ClientSocket.Active property to check if the socket is active.

1.8.1.3.1.13 TAstaClientSocket.ConnectionString

TAstaClientSocket

Declaration
```
function ConnectionString: string;
```

Description
Returns an English phrase to describe the current status of the socket in relation to an ASTA server, the Anchor server or a SOCKS server.

1.8.1.3.1.14 TAstaClientSocket.CursorOnQueries

TAstaClientSocket

Declaration
```
property CursorOnQueries: TAstaCursorActivity;
```

Description
Allows you to display "waiting" cursors while waiting for a result set -- see AstaClientDataSet. ShowQueryProgress.

1.8.1.3.1.15 TAstaClientSocket.DataSetCount

TAstaClientSocket

Declaration
```
property DataSetCount: Integer;
```

Description
Whenever AstaClientDataSets communicate with remote ASTA servers they use the AstaclientSocket and are assigned a unique identifier. Any AstaClientDataSet that has communciated with an ASTA server can be accessed in the DataSets property. The number of DataSets is available using the DataSetCount property. In this way, when Providers or other server side processes need to communicate with the AstaclientDataSet on the client, there is a path back to them from the server.

1.8.1.3.1.16 TAstaClientSocket.DataSets

TAstaClientSocket

Declaration
```
property DataSets[Index: Integer]: TDataSet;
```

Description
Whenever AstaClientDataSets communicate with remote ASTA servers they use the AstaClientSocket and are assigned a unique identifier. Any AstaClientDataSet that has communciated with an ASTA server can be accessed in the DataSets property. The number of DataSets is available using the DataSetCount property. In this way, when Providers or other server side processes need to communicate with the AstaClientDataSet on the client, there is a path back to them from the server.

1.8.1.3.1.17 TAstaClientSocket.DTPassword

TAstaClientSocket

Declaration
```
property DTPassword: string;
```

Description
ASTA provides powerful design time features by allowing a developer to access currently running Asta servers. This access does not require a password, a convenience that allows you to have easy access to your design time data without having to type in user names and passwords at every turn. It also opens up a potential security hole. Other ASTA developers, if they knew the IP address and the port of your AstaServer, would be able to gain access to your database if they were so inclined. One simple step that you can take to help avoid this problem is to switch your AstaServers from the default Port of 9000 to a different number above 1024.

To close the door completely, ASTA provides server-side and client-side properties that allow you to control who accesses the servers at design time. The AstaClientSocket and the AstaServerSocket both have DTUserName and DTPassword properties. Those properties are controlled by the AstaServerSocket's DTAccess property. DTAccess is defaulted to True. For maximum security, you should set DTAccess to false and then assign matching UserName and Passwords to the AstaClientSocket and AstaServerSocket DTUserName and DTPassword properties. If you set DTAccess to False, you must provide DTUserName and DTPassword values. Null values will not be accepted and the client will be terminated.

1.8.1.3.1.18 TAstaClientSocket.DTUserName

TAstaClientSocket

Declaration
```
property DTUserName: string;
```

Description
ASTA provides powerful design time features by allowing a developer to access currently running Asta servers. This access does not require a password, a convenience that allows you to have easy access to your design time data without having to type in user names

and passwords at every turn. It also opens up a potential security hole. Other ASTA developers, if they knew the IP address and the port of your ASTA server, would be able to gain access to your database if they were so inclined. One simple step that you can take to help avoid this problem is to switch your ASTA servers from the default port of 9000 to a different number above 1024.

To close the door completely, ASTA provides server-side and client-side properties that allow you to control who accesses the servers at design time. The AstaClientSocket and the AstaServerSocket both have DTUserName and DTPassword properties. Those properties are controlled by the AstaServerSocket's DTAccess property. DTAccess is defaulted to True. For maximum security, you should set DTAccess to false and then assign matching user name and password to the AstaClientSocket and AstaServerSocket DTUserName and DTPassword properties. If you set DTAccess to False, you must provide DTUserName and DTPassword values. Null values will not be accepted and the client will be terminated.

1.8.1.3.1.19 TAstaClientSocket.Encryption

TAstaClientSocket

Declaration
```
property Encryption: TEncryption;
```

Description
This property determines the encryption, if any, that will be employed by your application.

Value	Meaning
aeAstaEncrypt	Simple encryption -- set client and server properties.
aeNoEncryption	No encryption is used.
aeUserDefined	Your algorithm is used -- set client and server properties and the OnEncrypt and OnDecrypt event handlers for the client AND server.

NOTE: This property must have the SAME SETTING on the ServerSocket AND the ClientSocket.
NOTE: This property is used in conjunction with the Client's OnEncrypt and OnDecrypt events and the Server's corresponding OnEncrypt and OnDecrypt events.

1.8.1.3.1.20 TAstaClientSocket.Host

TAstaClientSocket

Declaration
```
property Host: string;
```

Description

The Host property accepts the domain name of the ASTA server (see detailed property desc). The Host or Address property must be set or the ASTA client will not connect. Use the Host property if you want to use the DNS system and enter addresses like "My.Company.Com", use the Address property if you wish to set IP addresses, like "204.200.220.25". Whether you use the Host or the Address property, the value entered must be the IP address of your target server.

NOTE: The Port property must also be set. If your server is at 204.200.120.2 and the ASTA server is running at Port 4000 then your client Address should be set to 204.200.120.2 and the Port should be set to 4000. If you set the Host and the Address, the Host property takes precedence.

See Address

1.8.1.3.1.21 TAstaClientSocket.KeepAlivePing

TAstaClientSocket

Declaration
```
property KeepAlivePing: Integer;
```

Description
On some dialup accounts uses will be disconnected after some period of inactivity. In the past, we have recommended that AstaClients call SendCodedMessage every couple of minutes using a timer. If your client is disconnected from an ASTA server it may be that the ISP is closing the connection after some period of inactivity. Use the KeepAlivePing:Integer property to set the number of minutes that the AstaclientSocket will send a empty string to the server to act as a "keep alive".

1.8.1.3.1.22 TAstaClientSocket.LoginMaxAttempts

TAstaClientSocket

Declaration
```
property LoginMaxAttempts: Integer;
```

Description
The number of attempts that a client can attempt to login to a server before the application terminates.

1.8.1.3.1.23 TAstaClientSocket.Password

TAstaClientSocket

Declaration
```
property Password: string;
```

Description
Specify a password to pass to the server. See AutoLoginDlg property.

1.8.1.3.1.24 TAstaClientSocket.Port

TAstaClientSocket

Declaration
```
property Port: Word;
```

Description
The Port property specifies the port that will be used in conjunction with the address or host property to create a "socket" connection to the AstaServer. A socket is the combination of an address and port. If your server is running at port 3500 and the server's address is 204.128.148.2 then you will fill in the ClientSocket Port and Address properties with those respective values.
Sometimes, in corporate environments, you may have to coordinate with the system administrator to ensure that TCP/IP traffic can pass through the port number that you would like to use.

1.8.1.3.1.25 TAstaClientSocket.ProgressBar

TAstaClientSocket

Declaration
```
property ProgressBar: Boolean;
```

Description
A progress bar can be used to give visual feedback to socket reads. If you would like the user to be able to see the progress of a large amount of data being read, turn on the ProgressBar property and then update your progress bar via the OnReadProgress event handler.
There is a slight performance penalty when using the OnReadProgress event, use it when the visual feedback is more important then the response time (it is more useful when receiving a large amount of data). The feature can be used selectively by toggling the ProgressBar property True or False.

1.8.1.3.1.26 TAstaClientSocket.SocksServerAddress

TAstaClientSocket

Declaration
```
property SocksServerAddress: string;
```

Description
Use this property to set the address of the SOCKS server you want to connect to through your ASTA server.

1.8.1.3.1.27 TAstaClientSocket.SocksServerPort

TAstaClientSocket

Declaration
```
property SocksServerPort: Word;
```

Description
Use this property to set the port of the SOCKS server usually 1080.

1.8.1.3.1.28 TAstaClientSocket.SocksUserName

TAstaClientSocket

Declaration
```
property SocksUserName: string;
```

Description

Use this property to set the user name that will be passed through to the SOCKS server you want to connect through to your ASTA server.

1.8.1.3.1.29 TAstaClientSocket.SQLDialect

TAstaClientSocket

Declaration
```
property SQLDialect: TSQLDialect;
type TSQLDialect = (sqlNone, sqlLocalSQL, sqlAccess, sqlMSSQLServer,
  sqlOracle, sqlSQLAnywhere, sqlInterbase, sqlDBISAM, sqlCustom);
```

Description
This will set the SQLOptions to a best fit for the target backend.

ASTA clients and servers can be configured a variety of ways to provide the SQL syntax used for your database. For a full discussion see SQL Generation.

1.8.1.3.1.30 TAstaClientSocket.SQLErrorHandling

TAstaClientSocket

Declaration
```
property SQLErrorHandling: TAstaSQLErrorSet;
```

Description
Allows you to specify how you would like to see SQL error messages.

Value	Meaning
seToStatusBar	Displays the errors in the middle panel of an AstaStatusBar.
seAsExceptions	Raise the SQL Error as an exception.

See the OnSQLError event handler if you are interested in working with the error strings.

1.8.1.3.1.31 TAstaClientSocket.SQLOptions

TAstaClientSocket

Declaration
property SQLOptions: TAstaServerSQLOptions;

Description
The SQLOptions property of the TAstaClientSocket on the client allows for full customization of the SQL that ASTA generates. The SQLOptions set is automatically filled if the UpdateSQLSyntax property is changed.

ASTA clients and servers can be configured a variety of ways to provide the SQL syntax used for your database. For a full discussion see SQL Generation.

1.8.1.3.1.32 TAstaClientSocket.SQLTransactionEnd

TAstaClientSocket

Declaration
```
property SQLTransactionEnd: string;
```

Description
Some databases allow transactions to be started and ended using SQL commands. Under ODBC, you can use Begin Transaction and COMMIT. If your database has these types of commands and you call AstaClientDataSet.ApplyUpdates(usmUseSQL) then the SQLTransactionStart and SQLTransactionEnd string properties will be used.

The preferred way of using transactions is to call AstaClientDataSet.ApplyUpdates(usmServerTransaction) which calls the AstaServerSocket.OnTransactionBegin event of the server to start a transaction and OnTransactionEnd to end a transaction.

1.8.1.3.1.33 TAstaClientSocket.SQLTransactionStart

TAstaClientSocket

Declaration
```
property SQLTransactionStart: string;
```

Description
Some databases allow transactions to be started and ended using SQL commands. Under ODBC, you can use Begin Transaction and COMMIT. If your database has these types of commands and you call AstaClientDataSet. ApplyUpdates(usmUseSQL) then the SQLTransactionStart and SQLTransactionEnd string properties will be used.

The preferred way of using transactions is to call AstaClientDataSet. ApplyUpdates(usmServerTransaction) which calls the AstaServerSocket.OnTransactionBeginevent of the server to start a transaction and OnTransactionEnd to end a transaction.

1.8.1.3.1.34 TAstaClientSocket.StatusBar

TAstaClientSocket

Declaration
```
property StatusBar: TStatusBar;
```

Description
Allows you to drop an AstaStatusBar component with basic connection status information. The AstaStatusBar will show SQL error messages when the SQLErrorHandling property is set to seToStatusBar.

1.8.1.3.1.35 TAstaClientSocket.UpdateSQLSyntax

TAstaClientSocket

Declaration
```
property UpdateSQLSyntax: TAstaUpdateSQL;
```

Description

There is no SQL standard for date and time syntax. UpdateSQLSyntax allows you to set set the way dates and datetime fields are represented in the SQL statments generated by ASTA.

Value	Meaning
usAccess	Handles date and time data according to Access syntax rules.
usBDE	Uses the BDE syntax for data and time data.
usODBC	For use with the ODBC demo server.
usODBCold	For use with versions of ODBCExpress prior to 5.02 when double colons where required when sending time values to a server.
usOracle	For use with Oracle.

See SQLGeneration for a full discussion on the SQL that ASTA can generate.

1.8.1.3.1.36 TAstaClientSocket.UserName

TAstaClientSocket

Declaration
```
property UserName: string;
```

Description
Used to specify a user name to pass to the server. See AutoLoginDlg property.

1.8.1.3.1.37 TAstaClientSocket.WebServer

TAstaClientSocket

Declaration
```
property WebServer: TAstaWebServer;
```

Description
ASTA supports HTTP clients using the AstaClientSocket. The AstaClientSocket.WebServer property is a property that allows the configuration and control of HTTP settings. The AstaClientSocket can optionally use the Microsoft WinInet.dll that comes with Internet Explorer. WinInet uses any registry settings defined for Internet Explorer which include any proxy and Socks settings. Setting the WebServer.WinInet property (Boolean) to true is the recommended way to traverse firewalls with ASTA. If Internet explorer can browser the Internet, your ASTA WinInet enabled client will be able to communicate with your remote ASTA server through a web server and Astahttp.dll

See also: Stateless Clients

1.8.1.3.2 Methods

TAstaClientSocket

CloseTheSocket
CommandLinePortCheck
ExpressWayDataSetSelect
FastConnect
FastConnectCombo
FieldIsRegisteredForTrigger
GetCodedParamList
Loaded
OpenTheSocket
RegisterTrigger
RequestUtilityInfo
SendAndBlock
SendAndBlockException
SendBlobMessage
SendChatEvent
SendChatPopup
SendCodedMessage
SendCodedParamList
SendCodedStream
SendGetCodedParamList
SendDataSetTransactions
SendMasterDetailAutoIncTransaction
SendMasterDetailOrderedTransactions
SendProviderTransactions
SetForIsapiUse
SetForProxyUse
SetupForSocks5Server
SocksConnect
TimerReconnect
UnRegisterTrigger
WaitingForServer

1.8.1.3.2.1 TAstaClientSocket.AddDataSet

TAstaClientSocket

Declaration
```
function AddDataSet(D: TDataSet): Integer;
```

Description
Adds a dataset to the AstaclientSocket.DataSetList and returns a unique identifier. Normally not called directly.

1.8.1.3.2.2 TAstaClientSocket.CloseTheSocket

TAstaClientSocket

Declaration
```
procedure CloseTheSocket;
```

Description
This sets the AstaClientSocket.Active to false and does an Application.ProcessMessage call until it is actually closed.

1.8.1.3.2.3 TAstaClientSocket.CommandLinePortcheck

TAstaClientSocket

Declaration
```
procedure CommandLinePortCheck;
```

Description
CommandLinePortCheck will respond to the following command line switches, none of which are case sensitive:

Value	Meaning
Port	Allows a port to be specified at run time. Example: Project1.exe Port=9001. Where 9001 is a number between 1 and 65535. See AstaClientSocket.Port for more details. The AstaServerSocket responds to this also.
Compression	Sets ASTA compression on. The AstaServerSocket responds to this also.
IPAddress	Allows an IP address to be defined at run time. Example: Project1.exe IPAddress=63.224.240.86.
Http	Sets the ASTA Protocol property to apHttpTunnel.

1.8.1.3.2.4 TAstaClientSocket.ExpressWayDataSetSelect

TAstaClientSocket

Declaration
```
procedure ExpressWayDataSetSelect(const DS: array of TDataSet);
```

Description
Allows multiple ASTAClientDataSet selects to be returned from an ASTA server in one trip to the server.

Example

```
AstaClientSocket1.ExpressWayDataSetSelect([AstaClientDataSet1,
  AstaClientDataSet2, AstaClientDataSet3]);
```

1.8.1.3.2.5 TAstaClientSocket.FastConnect

TAstaClientSocket

Declaration
```
procedure FastConnect;
```

Description

Used by the AstaClientSocket.ConnectionAction property to connect to a design time address. This is the same as setting the ConnectAction property to caUseDesignAddress.

1.8.1.3.2.6 TAstaClientSocket.FastConnectCombo

TAstaClientSocket

Declaration
```
procedure FastConnectCombo;
```

Description
This brings up a connection dialog box with IP address choices.

1.8.1.3.2.7 TAstaClientSocket.FieldIsRegisteredForTrigger

TAstaClientSocket

Declaration
```
function FieldIsRegisteredForTriggger(TableName, FieldName: string):
Boolean;
```

Description
Returns true if the TableName.FieldName has been previously registered with RegisterTrigger.

1.8.1.3.2.8 TAstaClientSocket.GetCodedParamList

TAstaClientSocket

Declaration
```
function GetCodedParamList(MsgID: Integer): TAstaParamList;
```

Description
GetCodedParamList allows you to use SendGetCodedParamList without having to send a ParamList to the server when you want to just return something from the server.

1.8.1.3.2.9 TAstaClientSocket.GetDataSet

TAstaClientSocket

Declaration
```
function GetDataSetID(DSID: Integer): TObject;
```

Description
Returns the TAstaClientDataSet for any id passed in. Used internally. See DataSetCount and DataSets.

1.8.1.3.2.10 TAstaClientSocket.HostToIPAddress

TAstaClientSocket

Declaration
```
function HostToIpAddress(const Host: string): string;
```

Description
DNS lookup returning ip address when passed in a host string.

1.8.1.3.2.11 TAstaClientSocket.IsBlocking

TAstaClientSocket

Declaration
```
function IsBlocking: Boolean;
```

Description
Returns True if the AstaClientSocket is setup for blocking calls.

1.8.1.3.2.12 TAstaClientSocket.IsHTTP

TAstaClientSocket

Declaration
```
function IsHTTP: Boolean;
```

Description
Returns True if the ClientSocket is running over HTTP.

1.8.1.3.2.13 TAstaClientSocket.IsStateless

TAstaClientSocket

Declaration
```
function IsStateless: Boolean;
```

Description
Returns True if the ClientSocket is running stateless (either blocking or HTTP).

1.8.1.3.2.14 TAstaClientSocket.Loaded

TAstaClientSocket

Declaration
```
procedure Loaded;
```

Description
The Loaded method triggers the ConnectionAction of the AstaClientSocket

1.8.1.3.2.15 TAstaClientSocket.OpenTheSocket

TAstaClientSocket

Declaration
```
procedure OpenTheSocket;
```

Description
This sets the AstaClientSocket.Active to true and does an Application.ProcessMessage call until the socket is actually active.

Early Connect

1.8.1.3.2.16 TAstaClientSocket.RegisterTrigger

TAstaClientSocket

Declaration
```
procedure RegisterTrigger(TableName, FieldName: string);
```

Description
ASTA alllows you to define default values that will be obtained from the server when SQL transactions are executed on the server from client side ApplyUpdate calls.Use the AstaClientSocket.RegisterTrigger method to register those Tablename.FieldNames that require a default value different for each client. These values must be supplied from the remote client or by the ASTA server so that they are available on the server from the AstaServerSocket.UserList and fired on the server from the AstaServerSocket.OnFireTrigger.

1.8.1.3.2.17 TAstaClientSocket.RequestUtilityInfo

TAstaClientSocket

Declaration
```
procedure RequestUtilityInfo(UtilInfo: TUtilInfoType; ParamString: string);
```

Description
Information about client EXEs that have been registered on ASTA servers for version updates can now be obtained from remote clients using the new RequestUtilityInfo call from the AstaClientSocket. This is an asyncronous request that will bring back a TAstaDataSet containing information about current client EXEs that have been registered on an ASTA server for the automatic client update feature.

You first call AstaClientSocket1.RequestUtilityInfo(uiAutoClientUpgradeInfo, ''), and a TAstaDataSet will be returned as part of the parameters to the AstaClientSocket.OnServerUtilityInfoEvent. These datasets are stored in the TAstaParamList as strings and can be converted to datasets using StringToDataSet (from astadrv2.pas), but you are then responsible for freeing the dataset.

Example

```
procedure TForm1.BitBtn1Click(Sender: TObject);
begin
  AstaClientSocket1.RequestUtilityInfo(uiAutoClientUpgradeInfo, '');
end;

procedure TForm1.AstaClientSocket1ServerUtilityInfoEvent(Sender:
  TObject;
  MsgID: integer; Params: TAstaParamList);
begin
  case MsgID of
    ord(uiAutoClientUpgradeInfo),
    ord(uiPublishedDirectories):
      begin
        DataSource1.DataSet.Free;
        DataSource1.DataSet := StringToDataSet(Params[0].AsString);
      end;
  end;
end;
```

```
procedure TForm1.FormDestroy(Sender: TObject);
begin
  DataSource1.DataSet.Free;
end;

procedure TForm1.BitBtn2Click(Sender: TObject);
var
  P: TAstaParamList;
begin
  if DataSource1.DataSet = nil then raise EDataBaseError.Create('You ' +
    'need to fetch a dataset first!');
  if DataSource1.DataSet.State in [dsEdit, dsInsert] then
    DataSource1.DataSet.Post;
  P := TAstaParamList.Create;
  P.FastAdd(DataSetToString(TAstaDataSet(DataSource1.DataSet)));
  AstaClientSocket1.RequestUtilityInfo(uiWriteAutoClientUpgradeDataSet,
    P.AsTokenizedString(False));
  P.Free;
end;

procedure TForm1.BitBtn3Click(Sender: TObject);
var
  M: TMemoryStream;
  P: TAstaParamList;
begin
  if not OpenDialog1.Execute then Exit;
  M := TMemoryStream.Create;
  M.LoadFromFile(OpenDialog1.FileName);
  P := TAstaParamList.Create;
  P.FastAdd('C:\Test.exe');
  //You have to know where you want to shove it to the server
  P.FastAdd('');
  P[1].AsStream := M;
  M.Free;
  AstaClientSocket1.RequestUtilityInfo(uiSaveClientUpgradeExe,
    P.AsTokenizedString(False));
  P.Free;
end;

procedure TForm1.BitBtn4Click(Sender: TObject);
begin
  AstaClientSocket1.RequestUtilityInfo(uiPublishedDirectories, '');
end;
```

1.8.1.3.2.18 TAstaClientSocket.SendAndBlock

TAstaClientSocket

Declaration

```
function SendAndBlock(TimeOut: Integer): string;
```

Description

The AstaClientSocket are by non-blocking event driven sockets. There are times however when you require blocking calls like within Isapi DLL's where is now windows message pump. In those situations you can use the SendAndBlock method after first setting the ClientType to ctblocking. This allows you to make any number of AstaclientDataSet calls and they will be cached until you cal the SendAndBlock method. Any errors will be returned in the String function result. To open a single AstaClientDataSet with a blocking

call use AstaclientDataSet1. OpenWithBlockingSocket(500,True). ASTA uses tcp/ip by default but can also run stateless using HTTP to defeat firewalls if necessary.

```
var
  Error: string;
begin
  AstaclientSocket1.Clienttype := ctblocking;
  AstaClientDataSet1.Open;
  AstaclientDataSet2.Open;
  Error:=AstaclientSocket1.SendAndBlock(1000);
  AstaClientDataSet1.SuitCaseData.Active := True;
  AstaclientDataSet2.SuitCaseData.Active := True;
  AstaClientDataSet1.Open;
  AstaclientDataSet2.Open;
end;
```

1.8.1.3.2.19 TAstaClientSocket.SendAndBlockException

TAstaClientSocket

Declaration
```
procedure SendAndBlockException(TimeOut: Integer);
```

Description
The AstaClientSocket are by non-blocking event driven sockets. There are times however when you require blocking calls like within Isapi DLL's where is now windows message pump. In those situations you can use the SendAndBlockException method after first setting the ClientType to ctblocking. This allows you to make any number of AstaclientDataSet calls and they will be cached until you cal the SendAndBlock method. Any errors will be returned in the String function result. To open a single AstaClientDataSet with a blocking call use AstaclientDataSet1.OpenWithBlockingSocket(500,True). ASTA uses tcp/ip by default but can also run stateless using HTTP to defeat firewalls if necessary. SendandBlockException raises an exception where SendAndBlock is a function that returns any error as the result.

1.8.1.3.2.20 TAstaClientSocket.SendBlobMessage

TAstaClientSocket

Declaration
```
procedure SendBlobMessage(TheClientStream: TMemoryStream);
```

Description

1.8.1.3.2.21 TAstaClientSocket.SendChatEvent

TAstaClientSocket

Declaration
```
procedure SendChatEvent(S: string);
```

Description

SendChatEvent broadcasts a message from the sending client to all other clients connected to the same server, but it is not intrusive.like SendChatPopup. This message must be intercepted and handled in the OnChatMessage event handler. If you do not assign the event handler, then the message will NOT be displayed.

This code broadcasts a message from the client to all the other clients.

```
procedure TForm1.Button2Click(Sender: TObject);
begin
  AstaClientSocket1.SendChatEvent(mChatMessage.Text);
end;
```

The following code writes the message to a TMemo component named MChatEvent. When the message is received at the client, the event is fired and the code below records the chat message in the memo.

```
procedure TForm1.AstaClientSocket1ChatMessage(Sender: TObject; S:
  string);
begin
  mChatEvent.Lines.Add(S);
  mChatEvent.Lines.Add(Chr(VK_Return));
end;
```

1.8.1.3.2.22 TAstaClientSocket.SendChatPopup

TAstaClientSocket

Declaration

```
procedure SendChatPopup(S: string);
```

Description

This method will broadcast a message from the sending client to all the other clients connected to the same server. This message is intrusive and it will stop the other clients from working while the message is displayed in a modal dialog that halts the other clients' progress. See SendChatEvent for an alternative.

The following code reads the input from a Memo named mChatMessage and broadcasts it to all the clients (including the sending client) in a popup dialog box.

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  AstaClientSocket1.SendChatPopup(mChatMessage.Text);
end;
```

1.8.1.3.2.23 TAstaClientSocket.SendCodedMessage

TAstaClientSocket

Declaration

```
procedure SendCodedMessage(ClientSocket: TCustomWinSocket;
  MsgID: Integer; Msg: string);
```

Description

SendCodedMessage allows you to send a message to the server, and then make a custom response back to the client. The SendCodedMessage method allows you to roll your own

protocol where certain MsgIDs have specific meaning to your application. The ability to create your own protocol in this fashion is a simple yet powerful technique. See the Simple Business Objects code below.

The simple code below sends a message to the server, but the message is sent with two different MsgIDs. The server will handle the message differently depending on which of the MsgIDs, 1700 or 1750, accompanies the message.

```
procedure TForm1.Button4Click(Sender: TObject);
begin
  AstaClientSocket1.SendCodedMessage(ClietnSocket, 1700,
    eClientCodedMessage.Text);
end;

procedure TForm1.Button3Click(Sender: TObject);
begin
  AstaClientSocket1.SendCodedMessage(ClientSocket, 1750,
    eClientCodedMessage.Text);
end;
```

The server responds wtih the following implementation code.

```
procedure TForm1.AstaServerSocket1CodedMessage(ClientSocket:
  TCustomWinSocket; Sender: TObject; MsgID: Integer; Msg: string);
begin
  case MsgID of
    1700: AstaServerSocket1.SendCodedMessage(ClientSocket,
            MsgID, UpperCase(Msg));
    1750: AstaServerSocket1.SendCodedMessage(Clientsocket,
            MsgID, LowerCase(Msg));
  end;
end;
```

When returned to the client, it is picked up in the AstaClientSocket's OnCodedMessage event.

Simple Business Objects

The following code shows how you could send CodedMessages to a business object instantiated on the server. In this scenario, 1800 and 1850 are "protocol codes" that you control. The client can call those codes, with or without a parameter and invoke a custom response from the server.

```
procedure TForm1.AstaServerSocket1CodedMessage(Sender: TObject;
  ClientSocket: TCustomWinSocket; MsgID: Integer; Msg: string);
begin
  case MsgID of
    1800: begin
            if MyBusinessObject.ChargeSalesTax(StrToInt(Msg)) then
              AstaServerSocket1.SendCodedMessage(ClientSocket,
                MsgID, 'TRUE')
            else
              AstaServerSocket1.SendCodedMessage(ClientSocket,
                MsgID, 'FALSE');
          end;
    1850: begin
            AstaServerSocket1.SendCodedMessage(ClientSocket, MsgID,
```

```
    MyBusinessObject.NewCreditLimit(Msg));
          end;
    end;
end;
```

1.8.1.3.2.24 TAstaClientSocket.SendCodedParamList

TAstaClientSocket

Declaration
```
procedure SendCodedParamList(MsgId: Integer; Params:TAstaParamList);
```

Description
The SendCodedParamList method is one of the more powerful methods in ASTA. It allows you to send mixed data types easily. For a fuller discussion see ASTA Messaging.

The following example is taken from the CodedParams Tutorial. It shows how to open a text file and send the file name, file size, the file itself and the current date and time to the server which then displays it in the ASTA server request memo that normally shows incoming SQL requests.

```
procedure TForm1.SendFileClick(Sender: TObject);
var
  Params: TAstaParamList;
begin
  if not OpenDialog1.Execute then Exit;
  Memo1.Lines.LoadFromFile(OpenDialog1.FileName);
  Params := TAstaParamList.Create;
  Params.FastAdd(OpenDialog1.FileName + ' File Size of ' +
    IntToStr(Length(Memo1.Text)));
  Params.FastAdd(Now);
  Params.FastAdd(Memo1.Text);
  AstaClientSocket1.SendCodedParamList(15, Params);
  Params.Free;
end;
```

The next bit of code shows how an AstaParamList can be created and packed with any kind of data including binary data. It uses an AstaParamList call "FastAdd" that adds an AstaParam, gives it a generic name, and sets the data type according to the type of data fed to it.

```
FastAdd('MyString');
FastAdd(45);
```

This is equivalent to saying:

```
Params.Add;
Params[0].AsString := 'My String';
Params.Add;
Params[1].AsInteger := 45;
```

1.8.1.3.2.25 TAstaClientSocket.SendCodedStream

TAstaClientSocket

Declaration

```
procedure SendCodedStream(MsgID: Integer; MS: TMemoryStream);
```

Description

The SendCodedStream compliments the SendCodedMessage method. It can be used to send streams to the server and then the server can respond with a custom action. The following code demonstrates the usage.

The client sends a message to the server.

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  AstaClientSocket1.SendCodedMessage(2000, '');
end;
```

The server responds by loading a memo into a stream and sending it to the client.

```
procedure TForm1.AstaServerSocket1CodedMessage(Sender: TObject;
  MsgID: Integer; Msg: string);
var
  MS : TMemoryStream;
begin
  case MsgID of
    2000: begin
            MS := TMemoryStream.Create;
            mSelectSend.Lines.SaveToStream(MS);
            AstaServerSocket1.SendCodedStream(2000, MS);
            MS.Free;
          end;
  end;
end;
```

The client receives the stream and displays it in a memo and saves it to a file.

```
procedure TForm1.AstaClientSocket1CodedStream(Sender: TObject;
  MsgID: Integer; MS: TMemoryStream);
begin
  case MsgID of
    2000: begin
            mFileFromServer.Lines.Clear;
            mFileFromServer.Lines.LoadFromStream(MS);
            mFileFromServer.Lines.SaveToFile('ATGTestFile.Txt');
          end;
  end;
end;
```

1.8.1.3.2.26 TAstaClientSocket.SendDataSetTransactions

TAstaClientSocket

Declaration

```
function SendDataSetTransactions(TransactionName: string; const DataSets:
  array of TDataSet): string;
```

Description
This is used in conjunction with CachedUpdates of the TAstaClientDataSet. SendDataSetTransactions allows you to send an ApplyUpdates command for any number of TAstaClientDataSets.

Example

```
SendDataSetTransaction('GeneralLedgerUpdate', [GLDataSet, ARDataSet,
  APDataSet, APDetailDataSet]);
```

1.8.1.3.2.27 TAstaClientSocket.SendDataSetTransactionsList

TAstaClientSocket

Declaration

```
function SendDataSetTransactionsList(TransactionName: string; DataSetList:
  TList; RaiseException: Boolean): string;
```

Description
Allows any number of AstaClientDataSets to be sent to the server in one transaction as a TList. For client side SQL only. Use SendProviderTransactions for Providers or ServerMethods. Transaction name is informational only.

1.8.1.3.2.28 TAstaClientSocket.SendGetCodeDBParamList

TAstaClientSocket

Declaration

```
function SendGetCodedDBParamList(MsgId: Integer; Params: TAstaParamList):
TAstaParamList;
```

Description
ASTA messaging has been used in more and more ways over the years and ASTA 3 introduces a fast and easy way to use ASTA messaging with the new call of SendGetCodedDBParamList.

```
procedure TForm1.Button1Click(Sender: TObject);
var
  pOut, pIn: TAstaParamList;
  i: Integer;
begin
  pOut := TAstaParamList.create;
  pIn := cs.SendGetCodedDBParamList(1000, POut);
  try
    for i := 0 to pIn.count - 1 do
      Memo1.Lines.Add(pIn[i].Name + ':' + pIn[i].AsString);
  finally
    pOut.Free;
    pIn.Free;
  end;
end;
```

See also: TAstaServerSocket.OnOutCodedDBParamList

1.8.1.3.2.29 TAstaClientSocket.SendGetCodedParamList

TAstaClientSocket

Declaration

```
function SendGetCodedParamList(MsgId: Integer; Params: TAstaParamList):
  TAstaParamList;
```

Description

The SendGetCodedParamList method is one of the more powerful methods in ASTA. It allows you to send mixed data types easily. SendGetCodedParamList will wait for the server to receive and send a reply and a TAstaParamList will be returned which you are responsible for destroying. SendGetCodedParamList is a blocking call. If you want to use asyncronous messaging the use SendCodedParamList. See ASTA Messaging for a fuller discussion on messaging options.

The following example shows how to open a text file and send the file name, file size, the file itself and the current date and time to the server which then displays it in the ASTA server request memo that normally shows incoming SQL requests and returns back a ParamList. On the server the OnCodedParamList event must be coded. On the server the OnCodedParamList event must be coded by:

1. Using any incoming params.
2. Clearing the params.
3. Filling the params with what you want to be returned to the client.
4. You need not send it back as it will be transported by ASTA in a thread.

```
procedure TForm1.SendFileClick(Sender: TObject);
var
  Params, RetParams: TAstaParamList;
begin
  if not OpenDialog1.Execute then Exit;
  Memo1.Lines.LoadFromFile(OpenDialog1.FileName);
  Params := TastaParamList.Create;
  try
    Params.FastAdd(OpenDialog1.FileName + ' File Size of ' +
      IntToStr(Length(Memo1.Text)));
    Params.FastAdd(Now);
    Params.FastAdd(Memo1.Text);
    RetParams := AstaClientSocket1.SendCodedParamList(15, Params);
  finally
    RetParams.Free;
    Params.Free;
  end;
end;
```

1.8.1.3.2.30 TAstaClientSocket.SendMasterDetailAutoIncTransaction

TAstaClientSocket

Declaration

```
function SendMasterDetailAutoIncTransaction(const DataSets: array of
  TDataSet): string;
```

Description

This method allows for Master/Detail transactions to be executed where the MasterDataSet uses an AutoIncrement field that is tagged to be refetched and the DetailDataSet will use this value as well. This means that in one server round trip a MasterDataset can insert a row, and the AutoIncrement value created on the server will

be used by the DetailDataSet.

There are some limitations:
1. EditMode must be set to Cached.
2. The MasterDataset can only handle the one row: the newly inserted row and it must be setup to have an AutoIncrement Field defined and tagged to be refetched.
3. No refetches are handled by the Detail since a new detail query will be fired by the server post anyhow.

Master Detail Support

1.8.1.3.2.31 TAstaClientSocket.SendMasterDetailOrderedTransactions

TAstaClientSocket

Declaration

```
function SendMasterDetailOrderedTransactions(TransactionName: string;
  MasterDataSet: TDataSet const  DetailDataSets: array of TDataSet;
  RaiseException: Boolean): string;
```

Description

In normal MasterDetail support, a new query is done for the detail dataset on each master change. In order to support fully deletes of detail and master you must use ASTA's disconnected master detail support so that all detail rows are fetched from the server.

In order to support the deletion of Master rows as well as detail rows, use SendMasterDetailOrderedTransactions so that the SQL will be generated in the order of:

1. Any Deletes from Detail DataSets.
2. Any Deletes from the Master DataSet
3. Master Update and Insert Statements
4. Detail Update and Insert Statements

Master Detail Support

1.8.1.3.2.32 TAstaClientSocket.SendNamedUserCodedParamList

TAstaClientSocket

Declaration

```
procedure SendNamedUserCodedParamList(TheUserName: string; MsgId: Integer;
  Params: TAstaParamList);
```

Description

1.8.1.3.2.33 TAstaClientSocket.SendProviderTransactions

TAstaClientSocket

Declaration
```
function SendProviderTransactions(TransactionName: string;
  const DataSets: array of TDataSet): string;
```

Description
This is used in conjunction with CachedUpdates of the TAstaClientDataSet which are set using the SQLGenerateLocation of the AstaClientDataSet. the EditMode property is NOT used with provider Cached Updates. SendProviderTransactions allows you to send an ApplyUpdates command for any number of TAstaClientDataSets that do NOT use SQL but use TAstaProviders or TAstaBusinessObjects on the server. The AstaClientDataSet.ExtraParams and AstaClientSocket.ClientSocketParams will also be transferred to the server side provider when using SendProviderTransactions.

Example

```
SendProviderTransactions('Provider GL Update', [GLDataSet, ARDataSet,
  APDataSet, APDetailDataSet]);
```

1.8.1.3.2.34 TAstaClientSocket.SendUserNameCodedParamList

TAstaClientSocket

Declaration
```
function SendUserNameCodedParamList(AUserName: string; MsgID: Integer;
  Params: TAstaParamList): TAstaParamList;
```

Description
Sends a CodedParamList to a named user. See IMA (Instant Messaging API).

1.8.1.3.2.35 TAstaClientSocket.ServerHasResponded

TAstaClientSocket

Declaration
```
procedure ServerHasResponded(Forceit: Boolean);
```

Description
Stops AstaSmartWait. This normally does not have to be called manually. When a request goes to the server ASTA goes into a wait state. When there is an error or exception on the server the wait must be stopped. ServerHasResponded allows you to programmtically stop the wait.

1.8.1.3.2.36 TAstaClientSocket.SetAESKeys

TAstaClientSocket

Declaration
```
procedure SetAESKeysString(const InKey, OutKey: string);
```

Description
AES encryption (currently US and Canada only to source code users) uses 2 keys. This call sets the InKey and OutKey. Note: the InKey on the server is the outkey to the client.

1.8.1.3.2.37 TAstaClientSocket.SetDESStringKey

TAstaClientSocket

Declaration
```
procedure SetDESStringKey(const AStringKey: string);
```

Description
Sets the Key used for DES encryption.

1.8.1.3.2.38 TAstaClientSocket.SetForIsapiUse

TAstaClientSocket

Declaration
```
procedure SetForIsapiUse(WebServerAddress, AstaServerAddress, AstaIsapiDll:
  string; WebServerPort, AstaServerPort: Word);
```

Description

1.8.1.3.2.39 TAstaClientSocket.SetForNormalTCPIP

TAstaClientSocket

Declaration
```
procedure SetForNormalTcpip(AstaServerAddress: string; AstaServerPort:
  Word);
```

Description
Sets the protocol to tcp/ip from HTTP.

1.8.1.3.2.40 TAstaClientSocket.SetForProxyUse

TAstaClientSocket

Declaration
```
procedure SetForProxyUse(AstaServerAddress, ProxyIPAddress: string;
  AstaServerPort, ProxyPort: Word);
```

Description

1.8.1.3.2.41 TAstaClientSocket.SetupForSocks5Server

TAstaClientSocket

Declaration
```
procedure SetupForSocks5Server(AstaServerAddress, TheSocks5ServerAddress,
  TheSocksUserName, TheSocksPassword: string; AstaServerPort,
  TheSocksServerPort: Word);
```

Description
Sets the client socket up for SOCKS 5 transport. See SOCKS Support.

1.8.1.3.2.42 TAstaClientSocket.SocksConnect

TAstaClientSocket

Declaration
```
procedure SocksConnect;
```

Description
Use SocksConnect to connect through a SOCKS server rather than using Active := True;

1.8.1.3.2.43 TAstaClientSocket.TimerReconnect

TAstaClientSocket

Declaration
```
procedure TimerReconnect(Delay: Integer)
```

Description
TimerReconnect uses an internal timer to trigger an Active := True call to reconnect to an ASTA server. You cannot set the Active property to true within the OnDisconnect event of the AstaClientSocket as the event must be allowed to complete. Use the TimerReconnect to trigger a reconnect to the server so that the active will execute after the OnDisconnect event has terminated. The Delay value is passed to the Timer as the Internal. TimerReconect is used internally when connecting to an ASTA Anchor Server for Load Balancing and Fail Over Support.

1.8.1.3.2.44 TAstaClientSocket.UnRegisterTrigger

TAstaClientSocket

Declaration
```
procedure UnRegisterTrigger(TableName,FieldName:String);
```

Description
ASTA alllows you to define default values that will be obtained from the server when SQL transactions are executed on the server from client side ApplyUpdate calls.Use the AstaClientSocket.RegisterTrigger method to register those Tablename.FieldNames that require a default value different for each client. UnRegisterTrigger removes any registered Tablename.FieldNames previously registered with RegisteredTrigger. To see if a Trigger exists call FieldIsRegisteredForTrigger.

1.8.1.3.2.45 TAstaClientSocket.WaitingForServer

TAstaClientSocket

Declaration
```
function WaitingForServer: Boolean;
```

Description
By nature, sockets are asyncronous. This means that a request is made and then a response gets fired to an event. Normal database development depends on a more procedural mode where you can write sequential lines of code. In order to "simulate" this procedural model, ASTA clients go into a "wait" state when requests are made from TAstaClientDataSets. You may check to see if the TAstaClientSocket is in this wait state by calling the WaitingForServer function.

1.8.1.3.2.46 TAstaClientSocket.WebServerCheck

TAstaClientSocket

Declaration
```
procedure WebServerCheck;
```

Description
Uses settings from AstaClientSocket.WebServer to set for HTTP if applicable.

1.8.1.3.2.47 TAstaClientSocket.WinINetActive

TAstaClientSocket

Declaration
```
function WinInetActive: Boolean;
```

Description
Returns True if the ClientSocket is running HTTP stateless with WinInet. Used internally.

1.8.1.3.3 Events

TAstaClientSocket

OnChatMessage
OnCodedMessage
OnCodedParamList
OnCodedStream
OnCompress
OnConnect
OnConnecting
OnConnectStatusChange
OnCustomConnect
OnCustomParamSyntax
OnCustomSQLSyntax
OnDecompress
OnDecrypt
OnDisconnect
OnEncrypt
OnError
OnLoginAttempt
OnLookup
OnRead
OnReadProgress
OnServerBroadcast
OnServerUtilityInfoEvent
OnSQLError
OnTerminateMessageFromServer
OnWrite

1.8.1.3.3.1 TAstaClientSocket.OnChatMessage

TAstaClientSocket

Declaration
```
property OnChatMessage: TAstaClientDataEvent;
```

Description
Add code to this event handler to when you wish to have your application respond to the SendChatEvent method.
The following code responds to a chat message by displaying the messag in a TMemo component named mChatEventLines.

```
procedure TForm1.AstaClientSocket1ChatMessage(Sender: TObject; S:
  string);
begin
  mChatEvent.Lines.Add(S);
  mChatEvent.Lines.Add(Chr(VK_Return));
end;
```

1.8.1.3.3.2 TAstaClientSocket.OnCodedMessage

TAstaClientSocket

Declaration
```
property OnCodedMessage: TAstaClientCodedDataEvent;
```

Description
The OnCodedMessage event handler allows you to take customized application action according to your own custom messages that have been sent via SendCodeMessage method.
The following code shows how the client socket might respond upon receiving certain messages.

```
procedure TForm1.AstaClientSocket1CodedMessage(Sender: TObject;
  MsgID: Integer; S: string);
begin
  case MsgID of
     500 : begin
             mSelectMessages.Lines.Add(S);
             PageControl1.ActivePage := tsOnCodedMessages;
           end;
     1700..1800 : eServerResponse.Text := S;
  end;
end;
```

1.8.1.3.3.3 TAstaClientSocket.OnCodedParamList

TAstaClientSocket

Declaration
```
property OnCodedParamList: TCodedParamEvent;
```

Description
The OnCodedParamList event handles messages sent with the SendCodedParamList method. Clients and servers have access to the SendCodedParamList method and the

OnCodedParamList event. This event allows you to retrieve the data that was sent. On the server, the OnCodedParamList event has the client socket passed to it while the client version of OnCodedParamList implicitly knows the server socket.
Here is the example code that comes with the AstaBDEServer example and the AstaODBCServer example. It corresponds to the code in the sample document in SendCodedParamList.

```
procedure TForm1.AstaServerSocket1CodedParamList(Sender: TObject;
  Clientsocket: TCustomWinSocket; MsgID: Integer; Params:
  TAstaParamList);
var
  I:Integer;
begin
  //if the client calls SendGetCodedParamList the Server would need to
  clear
  //the Params and add in anything that is needed to send back to the
  client.
  //ASTA will transport it back to the client.
  mRequests.Lines.Add('Coded Params for Msgid:' + IntToStr(Msgid));
  for I := 0 to Params.Count - 1 do
     mRequests.Lines.Add('  Params[' + IntToStr(i) + '] ->' +
       Params[i].AsString);
end;
```

When this code is called it parses each item that was sent and displays it in a memo.

1.8.1.3.3.4 TAstaClientSocket.OnCodedStream

TAstaClientSocket

Declaration
```
property OnCodedStream: TCodedStreamEvent;
```

Description
The OnCodedStream event handler is raised in response to a SendCodedStream message. In the example below, the client is responding to one of two SendCodedStream messages sent by the server. If the client receives a stream coded with MsgID 900 it loads the stream into a bitmap. If the client receives a 2000 in the MsgID it loads the stream into a memo and it launches the default editor associated with ".txt" extension.

```
procedure TForm1.AstaClientSocket1CodedStream(Sender: TObject;
  MsgID: Integer; MS: TMemoryStream);
begin
  case MsgID of
    900:  begin
            PageControl1.ActivePage := TabSheet4;
            Image2.Picture.Bitmap.LoadFromStream(MS);
          end;
    2000: begin
            mFileFromServer.Lines.Clear;
            mFileFromServer.Lines.LoadFromStream(MS);
            mFileFromServer.Lines.SaveToFile('ATGTestFile.Txt');
            if ShellExecute(0, 'open', 'ATGTestFile.txt', '', '',
              SW_SHOWNORMAL) <= 32 then
              ShowMessage('Unable to open text file.');
          end;
  end;
```

```
end;
```

1.8.1.3.3.5 TAstaClientSocket.OnCompress

TAstaClientSocket

Declaration
```
property OnCompress: TAstaClientDataStringEvent;
```

Description
This event handler allows you to add code to compress messages before they are sent across the network. It should be noted that compression/decompression routines will have performance considerations. Compressing large messages before they are sent over slow network connections is sensible. Compressing small messages before transmission over fast LANs may actually impede performance -- it might take longer to compress/decompress the data then it takes for the data to travel across the network. NOTE: If you add compression routines, you must add decompression routines. You must place your code in the appropriate even handlers (OnCompress and OnDecompress) of the AstaClientSocket and the AstaServerSocket.

1.8.1.3.3.6 TAstaClientSocket.OnConnect

TAstaClientSocket

Declaration
```
property OnConnect: TSocketNotifyEvent;
```

Description
See the Delphi help file for TClientSocket OnConnect. If you want to use any messaging or database methods this is the earliest you can call those methods as the socket needs to be connected to the server.

Note: You cannot open TAstaAstaClientDataSet in the FormCreate method as the socket may not be connected at that time!

1.8.1.3.3.7 TAstaClientSocket.OnConnecting

TAstaClientSocket

Declaration
```
property OnConnecting: TSocketNotifyEvent;
```

Description
Occurs for a client socket after the server socket has been located, but before the connection is established.

See the Delphi help file for TClientSocket OnConnecting.

1.8.1.3.3.8 TAstaClientSocket.OnConnectStatusChange

TAstaClientSocket

Declaration
```
property OnConnectStatusChange: TAstaClientStatechangeEvent;
```

Description

This event allows you to access the changing states of the client connection. Use this event to publish the connection state to the user interface if you don't want to use an AstaStatusBar.

1.8.1.3.3.9 TAstaClientSocket.OnCustomConnect

TAstaClientSocket

Declaration

```
property OnCustomConnect: TCustomConnectEvent;
```

Description

The OnCustomConnect event handler provides the opportunity for you to provide a custom login experience for your end users. This property is used in conjunction with the ConnectAction property.

A custom login provides you with the opportunity to present a promotional splash screen or even an advertising opportunity. If you select the caCustomConnect option of the AstaClientSocket's ConnectAction property, then you can customize a login screen. If you select this option, you can display your login screen from within the OnCustomConnectevent handler. In that event handler, you have access to the AstaClientSocket and all of its properties. If you elect to code this event handler, you must set the Address and Port properties or the Host and Port properties. Those properties become your responsibility when implementing a custom login.procedure.

```
procedure TForm1.AstaClientSocket1CustomConnect(Sender: TObject);
begin
  //CALL YOUR CUSTOM LOGIN SCREEN HERE
  //SET USERNAME, PASSWORD, AND OTHER PROPERTIES
  //YOU MUST SET THE ADDRESS (OR HOST) AND PORT
  AstaClientSocket1.Address := '206.210.19.131';
  AstaClientSocket1.Port := 8080;
end;
```

1.8.1.3.3.10 TAstaClientSocket.OnCustomParamSyntax

TAstaClientSocket

Declaration

```
type TSQLCustomParamsEvent = procedure(Sender: TObject;
  Item: TAstaParamItem; var TheResult: string; var Handled: Boolean);
```

Description

OnCustomParamSyntax allows you to customize the way that parameters are formatted in the Select statement when using parameterized queries. Remember to set the Handled argument to True when coding this event.

```
procedure TForm1.AstaClientSocket1CustomParamSyntax(Sender: TObject;
  Item: TAstaParamItem; var TheResult: string; var Handled: Boolean);
begin
  case Item.Datatype of
    ftDate: begin
              Handled := True;
              TheResult := QuotedStr(FormatDateTime('dd/mm/yyyy',
                Item.AsDateTime));
            end;
  end;
end;
```

See SQLGeneration for a full discussion of how ASTA generates SQL for insert, update and deletes.

1.8.1.3.3.11 TAstaClientSocket.OnCustomSQLError

TAstaClientSocket

Declaration
```
property OnCustomSQLError:
```

Description
Used in blocking socket support to trap error messages from a remote server.

1.8.1.3.3.12 TAstaClientSocket.OnCustomSQLSyntax

TAstaClientSocket

Declaration
```
type TSQLCustomFormatEvent = procedure(Sender: TObject; DS: TDataSet;
  FieldName: string; var TheResult: string; var Handled: Boolean) of
  object;
property OnCustomSQLSyntax: TSQLCustomFormatEvent;
```

Description
This event allows you to format any field to your specifications. Note that you MUST set the Handled argument to True if you are overriding the default format that ASTA uses. It is up to you to format the argument TheResult to your specifications.

```
procedure TForm1.AstaClientSocket1CustomSQLSyntax(Sender: TObject;
  DS: TDataSet; FieldName: string; var TheResult: string;
  var Handled: Boolean);
begin
    case DS.FieldByName(FieldName).DataType of
      ftDate: begin
                TheResult := QuotedStr(FormatDateTime('DD/MM/YYYY',
                  DS.FieldByName(FieldName).AsDateTime));
                  Handled := True;
              end
    end;
end;
```

1.8.1.3.3.13 TAstaClientSocket.OnDecompress

TAstaClientSocket

Declaration
```
property OnDecompress: TAstaClientDataStringEvent;
```

Description
This event handler allows you to add code to decompress messages after they have been sent, in an encrypted form, across the network. It should be noted that compression/decompression routines will have performance considerations. Compressing large messages before they are sent over slow network connections is sensible. Compressing small messages before transmission over fast LANs may actually impede performance -- it might take longer to compress/decompress than it takes the data to travel across the network.
Note: If you add compression routines, you must add decompression routines. You must

place your code in the appropriate even handlers (onCompress and OnDecompress) of the AstaClientSocket and the AstaServerSocket.

1.8.1.3.3.14 TAstaClientSocket.OnDecrypt

TAstaClientSocket

Declaration
```
property OnDecrypt: TAstaClientDataStringEvent;
```

Description
The OnDecrypt event handler allows you to add code for decrypting an encrypted message.
Note: If you add encryption routines to the OnEncrypt event handlers, then you must add decryption routines to the OnDecrypt event handlers. You must place your code in the appropriate even handlers (OnEncrypt and OnDecrypt) of the AstaClientSocket and the AstaServerSocket.

```
procedure TForm1.AstaClientSocket1Decrypt(Sender: TObject; var S:
  string);
begin
  S := MyDecryptRoutine(S);
end;
```

1.8.1.3.3.15 TAstaClientSocket.OnDisconnect

TAstaClientSocket

Declaration
```
property OnDisconnect: TSocketNotifyEvent;
```

Description
Occurs just before a client socket closes the connection to a server socket. See the Delphi help file for TClientSocket OnDisconnect.

If you do NOT code the OnDisconnect Event, an exception will be raised so it is recommended that you code the OnDisconnectEvent even with just a comment as ASTA will raise an exception if the event is not assigned.

Note:You cannot set the AstaClientSocket.Active to true in the OnDisconnect even but you must use a timer so that the setting of the Active property occurs after the event has fired. Use the TimerReconnect method from within the OnDisconnect event if you want to trigger a reconnect.

1.8.1.3.3.16 TAstaClientSocket.OnEncrypt

TAstaClientSocket

Declaration
```
property OnEncrypt: TAstaClientDataStringEvent;
```

Description
The OnEncrypt event handler allows you to add code for encrypting a message before transmitting it across a network.
Note: If you add encryption routines to the OnEncrypt event handlers, then you must add decryption routines to the OnDecrypt event handlers. You must place your code in the

appropriate even handlers (OnEncrypt and OnDecrypt) of the AstaClientSocket and the AstaServerSocket.

```
procedure TForm1.AstaClientSocket1Encrypt(Sender: TObject; var S:
  string);
begin
  S := MyEncryptRoutine(S);
end;
```

1.8.1.3.3.17 TAstaClientSocket.OnError

TAstaClientSocket

Declaration

```
property OnError: TSocketErrorEvent;
```

Description

This event allows you to trap for TCP errors occuring in your application. See the Delphi help file for TClientSocket OnErrorEvent.

1.8.1.3.3.18 TAstaClientSocket.OnLoginAttempt

TAstaClientSocket

Declaration

```
property OnLoginAttempt: TAstaLoginEvent;
type TAstaLoginEvent = procedure(Sender: TObject; ClientVerified: Boolean)
  of object;
```

Description

The OnLoginAttempt is used to control the login process at the client side of the connection. The event allows you control how many times the user is allowed to enter a user name and password combination. This is a good place to perform initialization as well.

The login process is controlled by a number of server and client side properties and events. The TAstaServerSocket has an OnClientAuthenticate event that controls whether or not the client has been accepted by the server.

```
procedure TForm1.AstaClientSocket1LoginAttempt(Sender: TObject;
  ClientVerified: Boolean);
begin
  //Track login attempts, kill app if they goof three times
  Inc(FLoginCounter);
  if (FLoginCounter >= 3) and (not ClientVerified) then
    Application.Terminate;
  else
  if ClientVerified then
    //Good login action goes here
    MainMenu1.Items[0].Visible := True
  else
    //Bad login action goes here -- reprompt login dialog
    AstaClientSocket1.AttemptClientLogin;
end;
```

1.8.1.3.3.19 TAstaClientSocket.OnLookup

TAstaClientSocket

Declaration
```
property OnLookup: TSocketNotifyEvent;
```

Description
Occurs when a client socket is about to look up the server socket with which it wants to connect. See the Delphi help file entry for TClientSocket.OnLookup event.

1.8.1.3.3.20 TAstaClientSocket.OnRead

TAstaClientSocket

Declaration
```
property OnRead: TSocketNotifyEvent;
```

Description
Occurs when a client socket should read information from the socket connection. See the Delphi help file entry for TAstaClientSocket.OnRead event.

1.8.1.3.3.21 TAstaClientSocket.OnReadProgress

TAstaClientSocket

Declaration
```
property OnReadProgress: TAstaSmartReadEvent;
```

Description
The OnReadProgress event handler allows you to assign values to a ProgressBar. Use it in conjunction with the ProgressBar property in order to provide the user with visual feedback on large data reads.
There is a slight performance penalty when using the OnReadProgress event, use it when the visual feedback is more important then the response time (it is more useful when receiving a large amount of data). The feature can be used selectively by toggling the ProgressBar property True or False.

```
procedure TForm1.AstaClientSocket1ReadProgress(Sender: TObject;
  Expected, Received: Integer);
begin
  ProgressBar1.Max := Expected;
  ProgressBar1.Position := Received;
end;
```

1.8.1.3.3.22 TAstaClientSocket.OnReceiveFieldDefs

TAstaClientSocket

Declaration
```
property OnReceiveFieldDefs: TAstaClientDataStringEvent;
```

Description

1.8.1.3.3.23 TAstaClientSocket.OnReceiveResultSet

TAstaClientSocket

Declaration
```
property OnReceiveResultSet: TAstaClientDataStringEvent;
```

Description

1.8.1.3.3.24 TAstaClientSocket.OnServerBroadcast

TAstaClientSocket

Declaration
```
property OnServerBroadcast: TAstaClientDataEvent;
```

Description
This event handler is raised when a message is sent from the server's SendBroadCast method. The server's SendBroadCastPopUp method displays client messages in a dialog. The SendBroadcast method allows you to choose how you would like to display the message.
When an OnServerBroadcase event is raised, the following code activates a specific page from a PageControl object and displays the message in a Memo field.

```
procedure TForm1.AstaClientSocket1ServerBroadcast(Sender: TObject;
  S: string);
const
  MessageSeparator = #13#10 + '================================' +
    #13#10;
begin
  PageControl1.ActivePage := tsOnBroadcast;
  mServerBroadcasts.Lines.Add(S);
  mServerBroadcasts.Lines.Add(MessageSeparator);
end;
```

1.8.1.3.3.25 TAstaClientSocket.OnServerUtilityInfoEvent

TAstaClientSocket

Declaration
```
property OnServerUtilityInfoEvent: TCodedParamEvent;
```

Description
This event receives a TAstaParamList after a request to RequestUtilityInfo
The following is an example of how to receive requests from an ASTA server in the OnServerUtilityInfoEvent.

```
procedure TAstaAnchorList.AnchorUtilInfo(Sender: TObject;
  MsgID: Integer; Params: TAstaParamList);
begin
  case Msgid of
    ord(uiUserCount): SetUserCount(Sender as TastaClientSocket,
      Params[0].AsInteger);
    ord(uiSessionInfo): SetGeneralInfo(Sender as
       TAstaClientSocket, Params[3].AsString);
```

```
end;
end;
```

1.8.1.3.3.26 TAstaClientSocket.OnSQLError

TAstaClientSocket

Declaration
```
property OnSQLError: TAstaClientDataEvent;
```

Description
The OnSQLError event handler provides you with access to the SQL Error returned from the database. The SQLErrorHandling property can be set to show the error as an exception or to the status bar, but in some environments it might be useful to write those errors to a log file or perhaps send them to an administrative application.

1.8.1.3.3.27 TAstaClientSocket.OnTerminateMessageFromServer

TAstaClientSocket

Declaration
```
property OnTerminateMessageFromServer: TAstaClientTerminatedEvent;
type TAstaClientTerminatedEvent = procedure (Sender: TObject;
  MsgFromServer: string var Handled: Boolean) of object;
```

Description
When a kill message is sent from an AstaServer the client application will terminate unless the OnTerminateMessageFromServer is coded and the var Handled:Boolean is set to true signifying that the event has been handled. For more secure servers don't send KillMessages but disconnect the remote client.

NOTE: the OnDisconnect event needs to be coded so that an exception is not raised.

Security Issues

1.8.1.3.3.28 TAstaClientSocket.OnWrite

TAstaClientSocket

Declaration
```
property OnWrite: TSocketNotifyEvent;
```

Description
Occurs when a client socket should write information to the socket connection. See the Delphi help file entry for TAstaClientSocket.OnWrite event.

1.8.1.4 TAsta2AuditDataSet

Properties : Methods

Unit
AstaDrv2

Declaration
```
type TAsta2AuditDataset = class(TAsta2CustomAuditDataSet)
```

Description
The Asta2AuditDataSet is an In memory dataset that can "track" changes allowing CancelUpdates and RevertRecord to be supported. It uses an internal AstaDataSet as the OldValuesDataSet so that any updates, inserts or deletes can be "tracked" or audited.

It is this dataset that TAstaClientDataSet inherits from. The Asta2AuditDataSet does not connect to a remote ASTA Server.

1.8.1.4.1 Properties

TAsta2AuditDataSet

OldValuesDataSet
StreamOldValuesDataSet
UpdateMethod

Derived from TAstaCustomDataSet
Aggregates
Indexes
IndexFieldCount
IndexFieldNames
IndexFields
IndexName
KeyExclusive
KeyFieldCount
MasterFields
MasterSource
ReadOnly
StreamOptions

Derived from TDataSet
Active
AggFields
AutoCalcFields
BlockReadSize
Bof
Bookmark
CanModify
Constraints
DataSetField
DataSource
DefaultFields
Designer
Eof

FieldCount
FieldDefList
FieldDefs
FieldList
Fields
FieldValues
Filter
Filtered
FilterOptions
Found
Modified
ObjectView
RecordCount
RecNo
RecordSize
SparseArrays
State

1.8.1.4.1.1 TAsta2AuditDataSet.OldValuesDataSet

TAsta2AuditDataSet

Declaration
```
property OldValuesDataSet: TAstaDataSet;
```

Description
TAsta2AuditDataSet maintain an internal FOldValuesDataSet which contains any deleted rows, newly inserted rows or the original values of any edited rows. This OldValuesDataSet allows support of CancelUpdates, RevertRecord and UpdatesPending.

1.8.1.4.1.2 TAsta2AuditDataSet.StreamOldValuesDataSet

TAsta2AuditDataSet

Declaration
```
property StreamOldValuesDataset: Boolean;
```

Description
Set StreamOldValuesDataSet to True if you want to save the OldValuesDataSet used to track deltas or changes to a dataset.

1.8.1.4.1.3 TAsta2AuditDataSet.UpdateMethod

TAsta2AuditDataSet

Declaration
```
property UpdateMethod: TAstaUpdateMethod;

type TAstaUpdateMethod = (umManual, umCached, umAfterPost);
```

Description

Determines when SQL is posted to the server. umAfterPost will fire after a record is posted. umCached requires the ApplyUpdates method to be called.

1.8.1.4.2 Methods

TAsta2AuditDataSet

CancelUpdates
EmptyCache
RevertRecord
TrackDeltas
UpdatesPending

Derived from TAstaCustomDataSet
AddBookmarkIndex
AddIndex
AddIndexFields
ApplyRange
CancelRange
CleanCloneFromDataSet
CloneCursor
CloneFieldsFromDataSet
CloneFieldsFromDataSetPreserveFields
CompareFields
DataTransfer
DefineSortOrder
EditKey
EditRangeEnd
EditRangeStart
Empty
FastFieldDefine
FilterCount
FindKey
FindNearest
GetRecordSize
GotoKey
GotoNearest
IsBlobField
LastNamedSort
LoadFromFile
LoadFromFileWithFields
LoadFromStream
LoadFromStreamwithFields
LoadFromString
NukeAllFieldInfo
RemoveSortOrder
SaveToFile
SaveToStream
SaveToString
SetKey
SetRange
SetRangeEnd
SetRangeStart

SortOrderSort
UnRegisterClone
ValidBookmark

Derived from TDataSet
ActiveBuffer
Append
AppendRecord
BookmarkValid
Cancel
CheckBrowseMode
ClearFields
Close
CompareBookmarks
ControlsDisabled
CreateBlobStream
CursorPosChanged
Delete
DisableControls
Edit
EnableControls
FieldByName
FindField
FindFirst
FindLast
FindNext
FindPrior
First
FreeBookmark
GetBlobFieldData
GetBookmark
GetCurrentRecord
GetDetailDataSets
GetDetailLinkFields
GetFieldData
GetFieldList
GetFieldNames
GotoBookmark
Insert
InsertRecord
IsEmpty
IsLinkedTo
IsSequenced
Last
Locate
Lookup
MoveBy
Next
Open
Post
Prior
Refresh
Resync

SetFields
Translate
UpdateCursorPos
UpdateRecord
UpdateStatus

1.8.1.4.2.1 TAsta2AuditDataSet.CancelUpdates

TAsta2AuditDataSet

Declaration
```
procedure CancelUpdates;
```

Description
When EditMode is set to anything other than Read Only; any edits, appends or deletes are recorded in a cache. Calling CancelUpdates rolls back any changes made, and puts the dataset back in the same state it was in when it was first opened, or after ApplyUpdates was called last, and clears the cache.

1.8.1.4.2.2 TAsta2AuditDataSet.EmptyCache

TAsta2AuditDataSet

Declaration
```
procedure EmptyCache;
```

Description
If EditMode is in Cached this will clear the OldValuesDataSet.

1.8.1.4.2.3 TAsta2AuditDataSet.RevertRecord

TAsta2AuditDataSet

Declaration
```
function RevertRecord: Boolean;
```

Description
If you make any change to specific row, and have already posted it, you can call RevertRecord to undo those changes if the UpdateMethod of the dataset has been set to umCached.

1.8.1.4.2.4 TAsta2AuditDataSet.UpdatesPending

TAsta2AuditDataSet

Declaration
```
function UpdatesPending: Boolean;
```

Description
Asta datasets maintain an internal OldValuesDataSet which contains any deleted rows, newly inserted rows or the original values of any edited rows. A call to UpdatesPending, while UpdateMethod is set to Cached, will check the OldValuesDataset and return True if any rows exist.

1.8.1.4.3 Events

TAsta2AuditDataSet

Derived from TAstaCustomDataSet
OnStreamEvent

Derived from TDataSet
AfterCancel
AfterClose
AfterDelete
AfterEdit
AfterInsert
AfterOpen
AfterPost
AfterScroll
BeforeCancel
BeforeClose
BeforeDelete
BeforeEdit
BeforeInsert
BeforeOpen
BeforePost
BeforeScroll
OnCalcFields
OnDeleteError
OnEditError
OnFilterRecord
OnNewRecord
OnPostError

1.8.1.5 TAstaCloneDataSet

Properties : Methods : Events

Unit
AstaCloneDataSet

Declaration
```
type TAstaCloneDataSet = class(TAsta2AuditDataSet)
```

Description
The TAstaCloneDataset allows for Fields and Data of any TDataSet to be "cloned" or copied at design time or runtime.

1.8.1.5.1 Properties

TAstaCloneDataSet

AddData
CallFirst
CloneDataSource
CloneIt
FieldsToSkip

Derived from TAsta2AuditDataSet
OldValuesDataSet
StreamOldValuesDataSet
UpdateMethod

Derived from TAstaCustomDataSet
Aggregates
Indexes
IndexFieldCount
IndexFieldNames
IndexFields
IndexName
KeyExclusive
KeyFieldCount
MasterFields
MasterSource
ReadOnly
StreamOptions

Derived from TDataSet
Active
AggFields
AutoCalcFields
BlockReadSize
Bof
Bookmark
CanModify
Constraints
DataSetField
DataSource
DefaultFields
Designer
Eof
FieldCount
FieldDefList
FieldDefs
FieldList
Fields
FieldValues
Filter
Filtered
FilterOptions
Found

Modified
ObjectView
RecordCount
RecNo
RecordSize
SparseArrays
State

1.8.1.5.1.1 TAstaCloneDataSet.AddData

TAstaCloneDataSet

Declaration
```
property AddData: Boolean;
```

Description

1.8.1.5.1.2 TAstaCloneDataSet.CallFirst

TAstaCloneDataSet

Declaration
```
property CallFirst: Boolean;
```

Description

1.8.1.5.1.3 TAstaCloneDataSet.CloneDataSource

TAstaCloneDataSet

Declaration
```
property CloneDataSource: TDataSet;
```

Description

1.8.1.5.1.4 TAstaCloneDataSet.CloneIt

TAstaCloneDataSet

Declaration
```
property CloneIt: Boolean;
```

Description

1.8.1.5.1.5 TAstaCloneDataSet.FieldsToSkip

TAstaCloneDataSet

Declaration
```
property FieldsToSkip: TStrings;
```

Description

1.8.1.5.2 Methods

TAstaCloneDataSet

Derived from TAsta2AuditDataSet
CancelUpdates
EmptyCache
RevertRecord
UpdatesPending

Derived from TAstaCustomDataSet
AddBookmarkIndex
AddIndex
AddIndexFields
ApplyRange
CancelRange
CleanCloneFromDataSet
CloneCursor
CloneFieldsFromDataSet
CloneFieldsFromDataSetPreserveFields
CompareFields
DataTransfer
DefineSortOrder
EditKey
EditRangeEnd
EditRangeStart
Empty
FastFieldDefine
FilterCount
FindKey
FindNearest
GetRecordSize
GotoKey
GotoNearest
IsBlobField
LastNamedSort
LoadFromFile
LoadFromFileWithFields
LoadFromStream
LoadFromStreamwithFields
LoadFromString
NukeAllFieldInfo
RemoveSortOrder
SaveToFile
SaveToStream
SaveToString
SetKey
SetRange
SetRangeEnd
SetRangeStart
SortOrderSort
UnRegisterClone
ValidBookmark

Derived from TDataSet

ActiveBuffer
Append
AppendRecord
BookmarkValid
Cancel
CheckBrowseMode
ClearFields
Close
CompareBookmarks
ControlsDisabled
CreateBlobStream
CursorPosChanged
Delete
DisableControls
Edit
EnableControls
FieldByName
FindField
FindFirst
FindLast
FindNext
FindPrior
First
FreeBookmark
GetBlobFieldData
GetBookmark
GetCurrentRecord
GetDetailDataSets
GetDetailLinkFields
GetFieldData
GetFieldList
GetFieldNames
GotoBookmark
Insert
InsertRecord
IsEmpty
IsLinkedTo
IsSequenced
Last
Locate
Lookup
MoveBy
Next
Open
Post
Prior
Refresh
Resync
SetFields
Translate
UpdateCursorPos
UpdateRecord

UpdateStatus

1.8.1.5.3 Events

TAstaCloneDataSet

Derived from TAstaCustomDataSet
OnStreamEvent

Derived from TDataSet
AfterCancel
AfterClose
AfterDelete
AfterEdit
AfterInsert
AfterOpen
AfterPost
AfterScroll
BeforeCancel
BeforeClose
BeforeDelete
BeforeEdit
BeforeInsert
BeforeOpen
BeforePost
BeforeScroll
OnCalcFields
OnDeleteError
OnEditError
OnFilterRecord
OnNewRecord
OnPostError

1.8.1.6 TAstaNestedDataSet

Properties : Methods : Events

Unit
AstaDrv2

Declaration
```
type TAstaNestedDataSet = class(TAsta2AuditDataSet)
```

Description
Use TAstaNestedDataSet to access data contained in a nested dataset. A nested dataset provides much of the functionality of a TAstaClientTable component, except that the data it accesses is stored in a nested table.

Set the DataSetField property to point to the persistent TField that represents the nested dataset to encapsulate.

1.8.1.6.1 Properties

TAstaNestedDataSet

ContentDefined
ParentDataSet
PrimeFields
UpdateTableName

Derived from TAsta2AuditDataSet
OldValuesDataSet
StreamOldValuesDataSet
UpdateMethod

Derived from TAstaCustomDataSet
Aggregates
Indexes
IndexFieldCount
IndexFieldNames
IndexFields
IndexName
KeyExclusive
KeyFieldCount
MasterFields
MasterSource
ReadOnly
StreamOptions

Derived from TDataSet
Active
AggFields
AutoCalcFields
BlockReadSize
Bof
Bookmark
CanModify
Constraints
DataSetField
DataSource
DefaultFields
Designer
Eof
FieldCount
FieldDefList
FieldDefs
FieldList
Fields
FieldValues
Filter
Filtered
FilterOptions
Found
Modified

ObjectView
RecordCount
RecNo
RecordSize
SparseArrays
State

1.8.1.6.1.1 TAstaNestedDataSet.ContentDefined

TAstaNestedDataSet

Declaration
```
property ContentDefined: Boolean;
```

Description

1.8.1.6.1.2 TAstaNestedDataSet.ParentDataSet

TAstaNestedDataSet

Declaration
```
property ParentDataSet: TAstaCustomDataset;
```

Description

1.8.1.6.1.3 TAstaNestedDataSet.PrimeFields

TAstaNestedDataSet

Declaration
```
property PrimeFields: TStrings;
```

Description
The PrimeFields property is used to provide information on the fields needed to fetch a row using a unique value. In order to have ASTA generate SQL automatically on edits, inserts and deletes, the PrimeFields must be defined. This is used by setting the UpdateMethod property to define auto SQL generation.

1.8.1.6.1.4 TAstaNestedDataSet.UpdateTableName

TAstaNestedDataSet

Declaration
```
property UpdateTableName: string;
```

Description
In order for ASTA to generate SQL for update, insert and delete statements, the UpdateTableName property must be set to the table that SQL updates should be applied to.

1.8.1.6.2 Methods

TAstaNestedDataSet

Derived from TAsta2AuditDataSet
CancelUpdates
EmptyCache
RevertRecord
UpdatesPending

Derived from TAstaCustomDataSet
AddBookmarkIndex
AddIndex
AddIndexFields
ApplyRange
CancelRange
CleanCloneFromDataSet
CloneCursor
CloneFieldsFromDataSet
CloneFieldsFromDataSetPreserveFields
CompareFields
DataTransfer
DefineSortOrder
EditKey
EditRangeEnd
EditRangeStart
Empty
FastFieldDefine
FilterCount
FindKey
FindNearest
GetRecordSize
GotoKey
GotoNearest
IsBlobField
LastNamedSort
LoadFromFile
LoadFromFileWithFields
LoadFromStream
LoadFromStreamwithFields
LoadFromString
NukeAllFieldInfo
RemoveSortOrder
SaveToFile
SaveToStream
SaveToString
SetKey
SetRange
SetRangeEnd
SetRangeStart
SortOrderSort
UnRegisterClone
ValidBookmark

Derived from TDataSet
ActiveBuffer
Append

AppendRecord
BookmarkValid
Cancel
CheckBrowseMode
ClearFields
Close
CompareBookmarks
ControlsDisabled
CreateBlobStream
CursorPosChanged
Delete
DisableControls
Edit
EnableControls
FieldByName
FindField
FindFirst
FindLast
FindNext
FindPrior
First
FreeBookmark
GetBlobFieldData
GetBookmark
GetCurrentRecord
GetDetailDataSets
GetDetailLinkFields
GetFieldData
GetFieldList
GetFieldNames
GotoBookmark
Insert
InsertRecord
IsEmpty
IsLinkedTo
IsSequenced
Last
Locate
Lookup
MoveBy
Next
Open
Post
Prior
Refresh
Resync
SetFields
Translate
UpdateCursorPos
UpdateRecord
UpdateStatus

1.8.1.6.3 Events

TAstaNestedDataSet

Derived from TAstaCustomDataSet
OnStreamEvent

Derived from TDataSet
AfterCancel
AfterClose
AfterDelete
AfterEdit
AfterInsert
AfterOpen
AfterPost
AfterScroll
BeforeCancel
BeforeClose
BeforeDelete
BeforeEdit
BeforeInsert
BeforeOpen
BeforePost
BeforeScroll
OnCalcFields
OnDeleteError
OnEditError
OnFilterRecord
OnNewRecord
OnPostError

1.8.1.7 TAstaUpdateSQL

Properties : Methods : Events

Unit
AstaUpdateSQL

Declaration
```
TAstaUpdateSQL = class(TComponent);
```

Description
Allows for the SQL that ASTA generatates to be supplemented.

1.8.1.7.1 Properties

TAstaUpdateSQL

DeleteSQL
InsertSQL
ModifySQL
Params

1.8.1.7.1.1 TAstaUpdateSQL.DeleteSQL

TAstaUpdateSQL

Declaration
```
property DeleteSQL: TStrings;
```

Description
Set DeleteSQL to the SQL DELETE statement to use when applying a deletion to a record. Statements can be parameterized queries. To create a DELETE statement at design time, use the UpdateSQL editor to create statements, such as:

DELETE FROM "country"

WHERE country = :country

At run time, an application can write a statement directly to this property to set or change the DELETE statement.

1.8.1.7.1.2 TAstaUpdateSQL.InsertSQL

TAstaUpdateSQL

Declaration
```
property InsertSQL: TStrings;
```

Description
Set InsertSQL to the SQL INSERT statement to use when applying an insertion to a dataset. Statements can be parameterized queries. To create a INSERT statement at design time, use the UpdateSQL editor to create statements, such as:

INSERT INTO "country"

(country, currency)

VALUES (:country, :currency)

At run time, an application can write a statement directly to this property to set or change the INSERT statement.

1.8.1.7.1.3 TAstaUpdateSQL.ModifySQL

TAstaUpdateSQL

Declaration
```
property ModifySQL: TStrings;
```

Description
Set ModifySQL to the SQL UPDATE statement to use when applying an updated record to a dataset. Statements can be parameterized queries. To create a UPDATE statement at design time, use the UpdateSQL editor to create statements, such as:

UPDATE "country"

SET country = :OLD_country, currency = :currency

At run time, an application can write a statement directly to this property to set or change the UPDATE statement.

Note: As the example illustrates, ModifySQL supports an extension to normal parameter binding. To retrieve the value of a field as it exists prior to application of cached updates, prefix the field name with 'OLD_'. This is especially useful when doing field comparisons in the WHERE clause of the statement.

1.8.1.7.1.4 TAstaUpdateSQL.Params

TAstaUpdateSQL

Declaration
```
property Params: TAstaParamList;
```

Description

1.8.1.7.2 Methods

TAstaUpdateSQL

GetSQL
ParseParams

1.8.1.7.2.1 TAstaUpdateSQL.GetSQL

TAstaUpdateSQL

Declaration
```
function GetSQL(UpdateKind: TDeltaType): TStrings;
```

Description
GetSQL is used to obtain the SQL statement for a specific type of update. GetSQL returns the SQL statement in the ModifySQL, InsertSQL, or DeleteSQL property, depending on the setting of the UpdateKind index.

UpdateKind can be any of the following:

Value	Meaning
dtEdit	Return the query text used to execute UPDATE statements (ModifySQL)
dtDelete	Return the query text used to execute DELETE statements (DeleteSQL)
dtAppend	Return the query text used to execute INSERT statements (InsertSQL)
dtAppendAndDelete	Return both the query text for INSERT and DELETE statements (InsertSQL and DeleteSQL)

1.8.1.7.2.2 TAstaUpdateSQL.ParseParams

TAstaUpdateSQL

Declaration
```
procedure ParseParams(TheParseSQL: string);
```

Description

1.8.1.7.3 Events

TAstaUpdateSQL

AfterSQLBatchInsert
AfterSQLItemInsert
BeforeSQLBatchInsert
BeforeSQLItemInsert

1.8.1.7.3.1 TAstaUpdateSQL.AfterSQLBatchInsert

TAstaUpdateSQL

Declaration
```
property AfterSQLBatchInsert: TAfterSQLBatchInsertEvent;

TAfterSQLBatchInsertEvent = procedure(Sender: TObject; var AddSQL: string;
  var AddParams: TAstaParamList) of object;
```

Description
After all SQL statements are generated, the application can insert extra SQL statements to send to the server. Only used for cached updates.

AddSQL contains the SQL statement to add to the list of SQL statements.
AddParams can be used to extend the AddSQL statement. Do NOT dispose AddParams.

1.8.1.7.3.2 TAstaUpdateSQL.AfterSQLItemInsert

TAstaUpdateSQL

Declaration
```
property AfterSQLItemInsert: TAfterSQLItemInsertEvent;

TAfterSQLItemInsertEvent = procedure(Sender: TObject; SQL: string; Params:
  TAstaParamList; DeltaType: TDeltaType;
```

Description
After every SQL item is generated, the application can insert an extra SQL statement to send to the server.

SQL contains the newly created SQL statement.
Params contain the newly created params.
DeltaType contains the type of SQL generated.
AddSQL contains the SQL statement to add to the list of SQL statements.

AddParams can be used to extend the AddSQL statement. Do NOT dispose AddParams.

1.8.1.7.3.3 TAstaUpdateSQL.BeforeSQLBatchInsert

TAstaUpdateSQL

Declaration
```
property BeforeSQLBatchInsert: TBeforeSQLBatchInsertEvent;

TBeforeSQLBatchInsertEvent = procedure(Sender: TObject; var AddSQL: string;
  var AddParams: TAstaParamList) of object;
```

Description
Before the SQL statement generatuion starts, the application can insert extra SQL statements to send to the server. Only used for cached updates.

AddSQL contains the SQL statement to add to the list of SQL statements.
AddParams can be used to extend the AddSQL statement. Do NOT dispose AddParams.

1.8.1.7.3.4 TAstaUpdateSQL.BeforeSQLItemInsert

TAstaUpdateSQL

Declaration
```
property BeforeSQLItemInsert: TBeforeSQLItemInsertEvent;

TBeforeSQLItemInsertEvent = procedure(Sender: TObject; var SQL: string; var
  Params: TAstaParamList; DeltaType: TDeltaType; var AddSQL: string; var
  AddParams: TAstaParamList) of object;
```

Description
Before each SQL item is generated, the application can insert an extra SQL statement to send to the server.

SQL contains the newly created SQL statement. SQL can be modified.
Params contain the newly created params. Params can be modified.
DeltaType contains the type of SQL generated.
AddSQL contains the SQL statement to add to the list of SQL statements.
AddParams can be used to extend the AddSQL statement. Do NOT dispose AddParams.

BeforeSQLItemInsert enables the application to modify the SQL statement and/or params which will be send to the server.

1.8.1.8 TAstaStatusBar

Unit
AstaStatusBar

Declaration
```
type TASTAStatusBar = class(TStatusBar)
```

Description
TAstaStatusBar is a descendent of Delphi's TStatusBar used to display connection status between your Asta client and server. To use it, place one on your form and make sure that the StatusBar property of your TAstaClientSocket points to this new component.

By default the TAstaStatusBar contains 3 panels, the first of which will show the Asta connection status messages and the third displays the text "Asta Technology Group". These panels can be modified at design time or run time by accessing the Panels property.

1.8.1.9 Remote Directory Components

The ASTA Remote Directory features allow you to "publish" a server directory. When a server publishes a directory, Asta clients can open files from and save files to that remote server directory. This is a convenient feature, but it has limitations and should be used with care. You should not publish a directory with thousands of files and dozens of subdirectories for performance reasons, and for security reasons you should not publish a root directory or other sensitive information. We recommend that you create a stand-alone directory dedicated to the sole task of publishing and accepting files.

Like many ASTA features, this one is a tandem implementation, you must coordinate actions on the server and the client sides or the feature will not operate properly.
To set up a server, you must drop a TAstaServerDirectory object onto the server application. Then you must assign the object to the AstaServerSocket.DirectoryPublisher property. Once that property is assigned, you can call the AstaServerSocket.PubishServerDirectories method.

The Server Directory Publisher dialog is raised by the PublishServerDirectories method. This dialog allows you to set several options; the directory, an alias name, and whether or not you want the directory to be automatically published every time the server starts up. In one examplet, I chose to publish the C:\eetemp directory. Since I didn't want the users to know the actual name of the server directory, I used the alias TestFiles. To acheive the same function in code without a UI control you can call PublishedDirectoriesFromRegistry and then UpdatePublishedDirectoriesToRegistry to save the contents back to the registry.The same calls can also be made by remote clients using the AstaclientSocket call RequestUtilityInfo.

Since this is a permanent feature of my application, I have selected the "Auto-Publish on Server Restart" checkbox. The information will be written to and automatically read from the registry automatically each time the server is started.
Now that the server side has been set up, you must attend to the client side. You need to add an AstaClientDirectory object to your form. The AstaClientDirectory.AstaClientSocket property should be assigned to the client's AstaClientSocket object
Then add an AstaOpenRemoteFile and/or and AstaSaveRemoteFile object to your project. The AstaClientDirectory properties of these objects should be set to the AstaClientDirectory1 object that you just previously added to the form. These objects will retrieve files from and save files to the server directories that are "published" at the server. The execute methods of the AstaOpenRemoteFile and the AstaSaveRemoteFile objects call up a select file dialog with the server's "published" directories.

1.8.2 Server Side

1.8.2.1 TAstaBusinessObjectManager

Properties : Methods

Unit
AstaMethodManager

Declaration
```
type TAstaBusinessObjectManager = class(TComponent);
```

Description
The TAstaBusinessObjectManager allows you to define server side Methods as TAstaActionItems with optional parameters that can then be used by ASTA clients using the TAstaClientDataSet.ServerMethod. This allows you to deploy ASTA client applications that require NO SQL on the client. See Server Side vs. Client Side SQL for a complete discussion. Each ServerMethod equires a TDataSet be attached to it either directly or through a TAstaProvider. The Method can have any number of Parameters defined which will be streamed down to the TAstaClientDataSet when a ServerMethod is chosen, either at design time or run time. When a TAstaClientDataSet is activated using a ServerMethod the Params are transported to the server and the OnAction event of the BusinessMethod is fired. It is your responsibility to use the incoming client params here for the TDataSet attached to the ServerMethod or for any other reason. To make a ServerMethod updatable, a TAstaProvider must be used.

1.8.2.1.1 Properties

TAstaBusinessObjectManager

Actions

1.8.2.1.1.1 TAstaBusinessObjectManager.Actions

Applies to
TAstaBusinessObjectsManager

Declaration
property Actions: TAstaActionItems;

Description
The Actions property holds a list of TAstaItems.

1.8.2.1.2 Methods

TAstabusinessObjectsManager

GetActionFromName

1.8.2.1.2.1 TAstaBusinessObjectsmanager.GetActionFromName

Applies to
TAstaBusinessObjectsManager
TAstaActionItem

Declaration
Function GetActionFromName(MethodName:String):TAstaActionItem;

Description
Each ServerMethod contains a pointer to the remote client socket that made the request. Use the ClientSocket property if you want to do any messaging from withing a ServerMethod.

1.8.2.1.3 TAstaActionItem

Properties : Events

Applies to
TAstaBusinessObjectsManager

Unit
AstaMethodManager

Declaration
```
type TAstaActionItem = class(TCollectionItem)
```

Description
TAstaActionItems are the actual methods of the TAstaBusinessObjectsManager and appear in TAstaClientDataSets as ServerMethods. Typically an ActionItem requires a TDataSet either through a TAstaProvider or any TDataSet descendent on the server. Set the UseNoDataSet property to true if you are not returning a result set but are only using the Params. SkyWire Clients can call servermethods through the SkyWire API.

SQL is not required for the Dataset of course and TTables can be used or just plain old TAstaDataSets (in memory). Any persistent field properties of the attached TDataset can be streamed down to the client if the SendFieldProperties property is set to True (default). This includes calculated fields.

Parameters can be defined for the TAstaActionItem and TAstaClientDataSets will see these parameters whenever the ServerMethod name is set, either at runtime or design time. If a TAstaProvider is used then the parameters of the dataset that the TAstaProvider is connected to can be used if the GetParamsFromProvider property is set to true. TAstaProviders can transmit their parameters if they are TParams OR the the Param Helper events are coded.

When a TAstaClientDataSet is opened using a server side method, the OnActionEvent of the TAstaActionItem is fired and the parameters from the TAstaClientDataSet are transmitted to the server. Any use of Params must be coded by you in the OnActionEvent either any setting of server side TDataSet.Params property or the updating of any returning parameters to the client of type ptOutput, ptResult, ptInputOutput.

1.8.2.1.3.1 Properties

TAstaActionItem

AstaProvider
ClientSocket
DataSet
DelayedProcess
GetParamsFromProvider
Method
Params
SendFieldProperties
ServerSocket
UseNoDataSet

TAstaActionItem

Declaration
```
property ServerSocket: TAstaServerSocket;
```

Description
Each ServerMethod contains a pointer to the AstaServerSocket. Use the ServertSocket property if you want to do any messaging from within a ServerMethod. By using the ServerSocket form the ServerMethod the method remains thread safe so that you do not have to access the actual form that the AstaServerSocket resides on.The Sender:TObject of the OnAction Event points to the ActionItem itself so it can be type cast to use the ServerSocket property and also to access the ClientSocket property.

TAstaActionItem

Declaration
```
property DelayedProcess: Boolean;
```

Description
When using the TAstaClientDataSet in ASTA client applications, the normal procedural character of database applications is preserved. Code is executed sequentially. The DelayedProcess property of a ServerMethod allows a server side method to be triggered and action then returning immediately to the TAstaclientDataSet so that other datasets or processing can be executed on the client while the server side method is executing. When the server side process is complete the dataset that is connected to the server method will be streamed back to the TAstaClientDataSet.

This is useful for triggering long running server side reports or processes and allow the client to continue with other work. Note: the ASTA Server must be running Threaded in order for this to work.

TAstaActionItem

Declaration
```
property ClientSocket: TCustomWinSocket;
```

Description
Each ServerMethod contains a pointer to the remote client socket that made the request. Use the ClientSocket property if you want to do any messaging from withing a ServerMethod. The Sender:TObject of the OnAction Event points to the ActionItem itself so it can be type cast to use the ClientSocket property or the ServerSocket property.

TAstaActionItem

Declaration
```
property UseNoDataSet: Boolean;
```

Description
Setting this property to true allows you to write a server side method that needs no dataset attached. ASTA will internally create an empty one row/one column dataset to return to the client.

TAstaActionItem

Declaration
```
property Params: TAstaParamList
```

Description
ServerMethods can have any number of Params defined. These params will be streamed back to AstaClientDataSets when a server method is choosen at runtime or design time. Params can be input or output params as defined in the TAstaParamList.

If a TAstaProvider is used, then the params from the Provider can be optionally used for the Params of the ServerMethod by setting the UseParamsFromProvider property to true.

TAstaActionItem

Declaration
```
property GetParamsFromProvider: Boolean;
```

Description
A server side method or ActionItem must be attached to a TDataSet. If theTAstaProviderproperty is set, then the TDataSet connected to the TAstaProvider will be used by the server method. If the attached Provider has params then this property optionally allows those params to be streamed back to the TAstaClientDataSet.

TAstaActionItem

Declaration
```
property Method: string;
```

Description
This is the name of the BusinessObject that will be seen on the TAstaClientDataSetServerMethod property.

TAstaActionItem

Declaration
```
property SendFieldProperties: Boolean;
```

Description
Set this property to true if you want to send extended field properties like DisplayWidth and DisplayLabel to AstaClientDataSets. This allows you also to add calculated fields on the server and send them to the client. By default, ServerDataSets will automatically send their Field properties to AstaClientDataSets.

TAstaActionItem

Declaration
```
property AstaProvider: TAstaProvider
```

Description
A server side method or ActionItem must be attached to a TDataSet. If theTAstaProviderproperty is set, then the TDataSet connected to the TAstaProvider will be used by the server method.

TAstaActionItem

Declaration
```
property DataSet: TDataSet
```

Description
A server side method or ActionItem must be attached to a TDataSet. If theTAstaProviderproperty is set, then the TDataSet connected to the TAstaProvider will be used by the server method.

A server method as defined by a TAstaActionItem must have a TDataSet connected to it which will be streamed back to the TAstaClientDataSet invoking the server method. The TAstaProvider, if connected will override the TDataSet property with the dataset that the TAstaProvider uses.

1.8.2.1.3.2 Events

TAstaActionItem

OnAction

TAstaActionItem

Declaration
```
type TAstaMethodActionEvent = procedure(Sender: TObject; ADataSet:
  TDataSet; ClientParams: TAstaParamList) of object;
```

Description
This event is triggered when a TAstaClientDataSet is opened on the client and ClientParams are streamed to the server. If the connected Dataset requires any of the client params it must be coded as well as updating any of the return params of type ptInputOutput, ptOutput or ptResult.

The Sender:TObject is the ActionItem itself and gives you access to the ServerSocket and ClientSocket if you need to send any messages from within the OnActionEvent.

Example

```
TAstaActionItem(Sender).ServerSocket.SendcodedMessage(TAstaActionItem(Se
  nder).ClientSocket, 1000, 'From a server method');
```

1.8.2.2 TAstaPDAServerPlugin

Properties : Methods : Events

Unit

Declaration

Description

1.8.2.2.1 Properties

1.8.2.2.1.1 TAstaPDAServerPlugin.AllowAnonymousPDAs

TAstaPDAServerPlugin

Declaration
```
property AllowAnonymousPDAs: Boolean;
```

Description

1.8.2.2.1.2 TAstaPDAServerPlugin.DatabasePlugin

TAstaPDAServerPlugin

Declaration
```
property DatabasePlugin: TAstaPdaDataBasePlugin;
```

Description

1.8.2.2.1.3 TAstaPDAServerPlugin.SkyWireAPI

TAstaPDAServerPlugin

Declaration
```
property SkyWireAPI: Boolean;
```

Description

1.8.2.2.1.4 TAstaPDAServerPlugin.SkyWireEmulation

TAstaPDAServerPlugin

Declaration
```
property SkyWireEmulation: Boolean;
```

Description

1.8.2.2.2 Methods

1.8.2.2.2.1 TAstaPDAServerPlugin.CreateDatabasePlugin

TAstaPDAServerPlugin

Declaration
```
procedure CreateDatabasePlugin;
```

Description

1.8.2.2.2.2 TAstaPDAServerPlugin.PluginProcessServerEngine

TAstaPDAServerPlugin

Declaration
```
function PluginProcessServerEngine(const TheClient: TUserRecord; var fsl:
  TAstaStringLine): Boolean; override;
```

Description

1.8.2.2.2.3 TAstaPDAServerPlugin.ProcessPdaParams

TAstaPDAServerPlugin

Declaration
```
function ProcessPdaParams(const TheClient: TUserRecord; Msgid: Integer;
  Params: TAstaParamList; var Handled: Boolean): Integer; override;
```

Description

1.8.2.2.3 Events

1.8.2.2.3.1 TAstaPDAServerPlugin.OnPalmSendDB

TAstaPDAServerPlugin

Declaration
```
property OnPalmSendDB: TAstaPDAReceiveDBEvent;
```

Description

1.8.2.2.3.2 TAstaPDAServerPlugin.OnPalmUpdateRequest

TAstaPDAServerPlugin

Declaration
```
property OnPalmUpdateRequest: TAstaServerPalmUpdateRequest;
```

Description

1.8.2.2.3.3 TAstaPDAServerPlugin.OnPDAAcquireRegToken

TAstaPDAServerPlugin

Declaration
```
property OnPDAAcquireRegToken: TAstaAcquireRegTokenEvent;
```

Description

1.8.2.2.3.4 TAstaPDAServerPlugin.OnPDAAuthentication

TAstaPDAServerPlugin

Declaration
```
property OnPDAAuthentication: TAstaAuthenticationEvent;
```

Description

1.8.2.2.3.5 TAstaPDAServerPlugin.OnPDAGetFileRequest

TAstaPDAServerPlugin

Declaration
```
property OnPDAGetFileRequest: TAstaPDAGetFileRequest;
```

Description

1.8.2.2.3.6 TAstaPDAServerPlugin.OnPDAParamList

TAstaPDAServerPlugin

Declaration
```
property OnPDAParamList: TAstaServerPDAParamListEvent;
type TAstaServerPDAParamListEvent = procedure(Sender: TObject; TheClient:
  TUserRecord; MsgID: integer; P: TAstaParamList) of object;
```

Description

1.8.2.2.3.7 TAstaPDAServerPlugin.OnPDAPasswordNeeded

TAstaPDAServerPlugin

Declaration
```
property OnPDAPasswordNeeded: TAstaPDAPasswordNeededEvent;
```

Description

1.8.2.2.3.8 TAstaPDAServerPlugin.OnPDARevokeRegToken

TAstaPDAServerPlugin

Declaration
```
property OnPDARevokeRegToken: TAstaRevokeRegTokenEvent;
```

Description

1.8.2.2.3.9 TAstaPDAServerPlugin.OnPDASendFile

TAstaPDAServerPlugin

Declaration

```
property OnPDASendFile: TAstaPDAReceiveDBEvent;
```

Description

1.8.2.2.3.10 TAstaPDAServerPlugin.OnPDAStream

TAstaPDAServerPlugin

Declaration
```
property OnPDAStream: TAstaServerPDAStreamEvent;
```

Description

1.8.2.2.3.11 TAstaPDAServerPlugin.OnPDAValidatePassword

TAstaPDAServerPlugin

Declaration
```
property OnPDAValidatePassword: TAstaPDAValidatePasswordEvent;
```

Description

1.8.2.2.3.12 TAstaPDAServerPlugin.OnPDAValidateRegToken

TAstaPDAServerPlugin

Declaration
```
property OnPDAValidateRegToken: TAstaPDAValidateRegTokenEvent;
```

Description

1.8.2.3 TAstaProvider

Properties : Events

Unit
AstaProvider

Declaration
```
type TAstaProvider = class(TComponent);
```

Description
The TAstaProvider is a server side component that allows ASTA clients to be deployed using NO SQL. Using a TAstaProvider and server side SQL generation, you will have access to each Insert, Update and Delete statement before they are to be executed on the server. This allows you to customize the SQL or even to ignore any rows that you choose not to update, insert or delete on the server. When using a TAstaClientDataSet connected to a Provider only Cached EditMode is supported. After Post editmode is not supported but posting each row change to the server can be simulated by calling ApplyUpdates in the AfterPost event. When an AstaClientDataSet's SQLGenerated location is changed to gsServer the AstaClientDataSet is automatically put in Cached Edit Mode. The Edit Mode property editor is not functional when using TAstaProviders.

TAstaProviders can also be used to monitor table changes on the server and those changes can be broadcasted to interested clients.

Any TAstaProviders present on ASTA server data modules can be seen by TAstaClientDataSets using the ProviderName property. When a ProviderName is chosen, any parameters present on the TAstaProvider will be streamed back to the client. If the TDataSet attached to the TAstaProvider is of type TParams this will happen automatically, otherwise you would have to code the Providers Param helper events. To support TAstaProviders, you must remember to register the datamodule(s) of your ASTA server.

1.8.2.3.1 Properties

TAstaProvider

Active
AstaServerSocket
AutoIncrementField
BroadCastsToOriginalClient
ClientDataSetExtraParams
ClientSocketParams
CompareCurrentServerValuesOnUpdates
Dataset
OracleSequence
Params
ParamsSupport
PrimeFields
RefetchOnUpdates
SendFieldProperties
ServerSideBroadcastFilter
UpdateTableName

1.8.2.3.1.1 TAstaProvider.Active

TAstaProvider

Declaration
```
property Active: Boolean;
```

Description
The Active property will get passed through to the dataset that is connected to the AstaProvider.

1.8.2.3.1.2 TAstaProvider.AstaServerSocket

TAstaProvider

Declaration
```
property AstaServerSocket: TObject;
```

Description
This provides a pointer back to the AstaServerSocket in case you want to make any thread safe ASTA messaging calls using the TAstaProvider on the server DataModule. The property can be type cast back to a TAstaServerSocket.

Example

```
TAstaServerSocket(MyProvider).AstaServerSocket.SendCodedMessage(ClientSo
  cket, 100, 'Hello From the Provider');
```

1.8.2.3.1.3 TAstaProvider.BroadCastsToOriginalClient

TAstaProvider

Declaration
```
property BroadCastsToOriginalClient: Boolean;
```

Description
When an AstaClientDataSet is set to receive Provider Broadcasts, this property will determine whether inserts, updates or deletes from the originating client will be broadcast back to the original client. The default is False.

Provider BroadCasts

1.8.2.3.1.4 TAstaProvider.ClientDataSetExtraParams

TAstaProvider

Declaration
```
property ClientDataSetExtraParams: TAstaParamList;
```

Description
The AstaClientDataSet has a public ExtraParams:TAstaParamList property that will be transfered to server side TAstaProviders with any call to SendProviderTransactions along with the AstaclientSocket.ClientSocketParams which will appear on the AstaProvider.ClientSocketParams.

1.8.2.3.1.5 TAstaProvider.ClientSocketParams

TAstaProvider

Declaration
```
property ClientSocketParams: TAstaParamList;
```

Description
The AstaClientDataSet has a public ClientSocketParams:TAstaParamListproperty that will be transfered to server side TAstaProviders with any call to SendProviderTransactions that will contain anything in the AstaClientSocket.ClientSocketParams along with the AstaclientDataSet.ExtraParams which will appear on the AstaProvider.ClientDataSetExtraParams. This allows you to transfer disparate information to AstaProviders.

1.8.2.3.1.6 TAstaProvider.CompareCurrentServerValuesOnUpdates

TAstaProvider

Declaration
```
property CompareServerValuesOnUpdates: Boolean;
```

Description
If set to True the CompareServerValuesOnUpdates will do a server side select before the BeforeUpdate is fired so that the ServerDataSet of the OnBeforeupdate is populated with current values from the server.

1.8.2.3.1.7 TAstaProvider.Dataset

TAstaProvider

Declaration
```
property DataSet: TDataSet;
```

Description
AstaProviders require a TDataSet descendent whose contents will be streamed back to TAstaClientDataSets.

1.8.2.3.1.8 TAstaProvider.Params

TAstaProvider

Declaration
```
property Params: TAstaParamList;
```

Description
The datasets attached to AstaProviders usually provide a Params property in order to support parameterized queries. If that property is of type TParams, these parameters will be automatically streamed back to the TAstaClientDataSet pointing toward the AstaProvider.

Often however, the TDataSet descendant used on an ASTA server will not use a TParams property but have their own class similar to TParams. ASTA must be able to transalate any Params to a TAstaParamList so it can be safely streamed across process boundries. To do this there are Param Helper Events on the TAstaProvider.

1.8.2.3.1.9 TAstaProvider.ParamsSupport

TAstaProvider

Declaration

Description
TAstaProviders require a TDataSet be attached to it. If that TDataSet has a Params property that is type TParams, these params will be automatically streamed back to TAstaClientDataSets and be populated from the client before the provider's TDataSet is opened.

In order to support all the unknown Params that are used on third party TDataSet

descendents, the TAstaServerSocket has two methods that require coding in order to achieve transparent and complete Param support for the TAstaClientDataSet.

OnProviderConvertParams
OnProviderSetParams

1.8.2.3.1.10 TAstaProvider.PrimeFields

TAstaProvider

Declaration
```
property PrimeFields: TStrings;
```

Description
The PrimeFields property is used to provide information on the fields needed to fetch a row using a unique value. In order to be able to have ASTA generate SQL automatically on edits, inserts and deletes the PrimeFields must be defined.

Related Topics
ASTA SQL Generation

1.8.2.3.1.11 TAstaProvider.RefetchOnUpdates

TAstaProvider

Declaration
```
property RefetchOnUpdates: Boolean;
```

Description
The RefetchOnInsert property allows you to easily retrieve specific field values after an Insert or Update statement. If you have an auto increment field or perhaps a timestamp field that is inserted by a trigger, you can use this property to return those values. This property will work for a single transaction or a group of transactions.

Specify the fields that you want to retrieve in the RefetchOnInsert property editor. Whenever an Insert statement is fired, the values will be returned to the client. The AfterRefetchOnInsert event will notify you when the values have been returned. Code that event if you need to take immediate action with the returned value.

If an auto increment field is defined and included in this list of fields, then you must code the OnInsertAutoIncrementFetch event on the TAstaServerSocket. For Paradox and Access a call to Max(FieldName) is made. For SQLServer and SQL Anywhere you can code a call to the variable @@identity to retrieve the last autoincrement value. The ASTA server then uses this value, or the prime key fields values to fetch all of the fields as specified in the RefetchOnInsert property. All of the server SQL is executed and when it is completed, the RefetchOnInsert values are streamed from the ASTA server and matched up to the appropriate rows in the TAstaClientDataSet.

Note: this property will not work if your database does not support transactions.

Refetch Discussion

1.8.2.3.1.12 TAstaProvider.RetainCurrentValuesDataSet

TAstaProvider

Declaration
```
property RetainCurrentValuesDataSet: Boolean;
```

Description
Set RetainCurrentValuesDataSet to True if you want the TAstaProvider to hold onto or maintain the CurrentValuesDataSet presented in the OnBeforeUpdate and onBeforeInsert events. this can be useful in manual broadcasts or if you want to code the OnProviderCompleteTransationEvent.

1.8.2.3.1.13 TAstaProvider.SendFieldProperties

TAstaProvider

Declaration
```
property SendFieldProperties: Boolean;
```

Description
Set this property to true if you want to send extended field properties like DisplayWidth and DisplayLabel to AstaClientDataSets. This allows you also to add calculated fields on the server and send them to the client along with other extended field properties. There is a cost do doing this. By default this is set to true. This means that persistent fields for remote AstaClientDataSets are defined on the server if SendFieldProperties is set to true. If you want to define persistent field properities on the client, set SendFieldProperties to false.

See also
TAstaActionItem.SendFieldProperties
TAstaServerSocket.PersistentFieldTranslateEvent

1.8.2.3.1.14 TAstaProvider.ServerSideBroadcastFilter

TAstaProvider

Declaration
```
property ServerSideBroadcastFilter: string;
```

Description
Broadcasts can be an expensive operation so by adding the ability to filter any datasets for broadcast can cut down on the amount of broadcasts. ASTA providers both client and server side filtering. The ServerSideBroadFilter, which supports all the expressions in the TDataSet Filter:String property, allows for broadcasts to be only made on certain conditions. combined with the AstaclientDataSet.ProviderFilter:String property broadcasts can be customized to be made extremely efficient and targetted only specific data changes.

1.8.2.3.1.15 TAstaProvider.UpdateTableName

TAstaProvider

Declaration
```
property UpdateTableName: string;
```

Description
In order for ASTA to generate SQL on update, insert and deletes, the UpdateTableName property must be set. See Asta SQL Generation for a full discussion.

1.8.2.3.2 Events

TAstaProvider

AfterInsert
AfterDelete
AfterRefetch
AfterUpate
BeforeInsert
BeforeDelete
BeforeOpen
BeforeUpdate
OnAfterTransaction
OnBeforeTransaction
OnBroadCastDecideEvent
OnRefetchDataSet
OnRefetchSQL

1.8.2.3.2.1 TAstaProvider.AfterOpen

TAstaProvider

Declaration
```
property AfterOpen: TAstaProviderOpenEvent;
type TAstaProviderOpenEvent = procedure(Sender: TObject; ClientSocket:
  TCustomWinsocket; ClientParams: TAstaParamList) of object;
```

Description
Fires after a TAstaProvider is opened.

1.8.2.3.2.2 TAstaProvider.AfterDelete

TAstaProvider

Declaration
```
property AfterDelete: TAstaAfterDeleteEvent;
type TAstaAfterDeleteEvent = procedure(Sender: TObject; ClientSocket:
  TCustomWinSocket; ExecQuery: TComponent; OrigianlValueDataSet,
  ServerValueDataSet: TDataset) of object;
```

Description
This event will fire after a delete using server side SQL and allows you to execute additional SQL statements using the scratch ExecQuery that is passed through by this

event. The OrginalValueDataSet will contain will contain the values from the client dataset and the ServerValueDataset will contain current Values on the server before the BeforeDelete event was fired.

1.8.2.3.2.3 TAstaProvider.AfterInsert

TAstaProvider

Declaration

```
property AfterInsert: TAstaInsertEvent;
type TAstaAfterInsertEvent = procedure(Sender: TObject; ClientSocket:
  TCustomWinSocket; ExecQuery: TComponent; CurrentValueDataSet: TDataset)
  of object;
```

Description

This event will fire after an insert using server side SQL and allows you to execute additional SQL statements using the scratch ExecQuery that is passed through by this event. The CurrentValueDataSet will contain the values from the client dataset.

1.8.2.3.2.4 TAstaProvider.AfterRefetch

TAstaProvider

Declaration

```
property AfterRefetch: TAstaProviderAfterRefetchEvent;
type TAstaProviderAfterRefetch = procedure(Sender: TObject; ClientSocket:
  TCustomWinSocket; RefetchedDataSet, CurrentValuesDataSet: TDataSet) of
  object;
```

Description

This event will fire after a refetch using server side SQL. The CurrentValueDataSet will contain the values from the client dataset and the RefetchedDataSet contain the values just refetched.

1.8.2.3.2.5 TAstaProvider.AfterUpdate

TAstaProvider

Declaration

```
property AfterUpdate: TAstaAfterUpdate;
type TAstaAfterUpdateEvent = procedure(Sender: TObject; ClientSocket:
  TCustomWinSocket; ExecQuery: TComponent; CurrentValueDataSet: TDataset)
  of object;
```

Description

This event will fire after an update using server side SQLand allows you to execute additional SQL statements using the scratch ExecQuery that is passed through by this event. The CurrentValueDataSet will contain the values from the client dataset.

1.8.2.3.2.6 TAstaProvider.BeforeDelete

TAstaProvider

Declaration

```
property AfterInsert: TAstaDeleteEvent;
type TAstaDeleteEvent = procedure(Sender: TObject;
  ClientSocket: TCustomWinSocket; ExecQuery: Component;
```

```
OriginalValueDataSet, ServerValueDataSet: TDataset;
  var Handled: Boolean) of object;
```

Description
This event will fire before a delete using server side SQLand allows you to execute additional SQL statements using the scratch ExecQuery that is passed through by this event. The OriginalValueDataSet will contain the values from the client dataset. If you do NOT want ASTA to fire the Delete statement then set Handled to True.
(ServerValueDataSet has not been implemented yet but would contain the current values for the prime key fields).

1.8.2.3.2.7 TAstaProvider.BeforeInsert

Applies to
TAstaProvider

Declaration
```
type TAstaInsertEvent = procedure(Sender: TObject;
  ClientSocket: TCustomWinSocket; ExecQuery: TComponent;
  CurrentValueDataSet: TDataset; var Handled: Boolean) of object;
property AfterInsert: TAstaInsertEvent;
```

Description
This event will fire before an insert using server side SQLand allows you to execute additional SQL statements using the scratch ExecQuery that is passed through by this event. The CurrentValueDataSet will contain the values from the client dataset. If you do NOT want ASTA to generate any SQL and execute it for you then set the Handled argument to True.

1.8.2.3.2.8 TAstaProvider.BeforeOpen

Applies to
TAstaProvider

Declaration
```
type TAstaProviderOpenEvent = procedure(Sender: TObject;
  ClientSocket: TCustomWinsocket;
  ClientParams: TAstaParamList) of object
property BeforeOpen: TAstaProviderOpenEvent;
```

Description
This event will fire before the dataset attached to the TAstaProvider is opened and will contain the ClientParams streamed in from the TAstaclientDataSet.Params property.

1.8.2.3.2.9 TAstaProvider.BeforeUpdate

Applies to
TAstaProvider

Declaration
```
type
TAstaUpdateEvent = procedure(Sender: TObject; ClientSocket:
TCustomWinSocket; ExecQuery: TComponent;
  OriginalValueDataSet,CurrentValueDataSet,ServerValueDatASet;CurrentValueD
ataSet: TDataset;         var Handled: Boolean) of object;
property BeforeUpdate: TAstaUpdateEvent;
```

Description
This event will fire before an update using server side SQLand allows you to execute additional SQL statements using the scratch ExecQuery that is passed through by this event. The CurrentValueDataSet will contain the current values from the client dataset, the originalValueDataSet will contain the original values before the update and if the CompareCurrentServerValuesOnUpdates is set to true the ServerValueDataSet will contain freshly populate data from the server. If the var Handled:Boolean is set to true the AstaProvider will not generate any SQL. Use the passed in ExecQuery by typecasting it to fire any Update SQL in this event.

1.8.2.3.2.10 TAstaProvider.AfterTransaction

TAstaProvider

Declaration
```
property OnAfterTransaction: TAstaProviderAfterTransactionEvent;
type TAstaProviderAfterTransactionEvent = procedure(Sender: TObject;
  ClientSocket: TCustomWinSocket; ExecQuery: TComponent;
  CurrentValueDataSet: TDataset; TransactionFailed: Boolean) of object;
```

Description
The OnAfterTransaction event is fired for a provider after a transaction has finihsed. A transaction will only be used if ApplyUpdates is called at the client side with usmServerTransaction.

The CurrentValueDataSet is the values as they are now in the database (ie does not include deleted rows). The success or otherwise of the transaction can be seen in the TransactionFailed parameter.

You can use the passed in ExecQuery to execute some other database action if you wish. The type of ExecQuery is the same as the object returned from ThreadedDBSupplyQuery with DBAction=ttExec.

1.8.2.3.2.11 TAstaProvider.BeforeTransaction

TAstaProvider

Declaration
```
property OnBeforeTransaction: TAstaProviderBeforeTransactionEvent;
type TAstaProviderBeforeTransactionEvent = procedure(Sender: TObject;
  ClientSocket: TCustomWinSocket; ExecQuery: TComponent; OrigValueDataSet,
  CurrentValueDataSet: TDataset; var Handled: Boolean) of object;
```

Description
The OnBeforeTransaction event will be fired for a provider before a transaction is started. A transaction will only be used if ApplyUpdates is called at the client side with usmServerTransaction.

If you wish to handle the transaction yourself, set Handled to True.

CurrentValueDataSet is the values from the client and does not include deleted rows. OrigValueDataSet is the old values dataset from the client and includes deleted rows but not appended rows.

1.8.2.3.2.12 TAstaProvider.OnBroadCastDecideEvent

TAstaProvider

Declaration

```
type BroadCastOkEvent = procedure(Sender: TObject; NewValueDataset:
  TDataSet; var BroadCastOK: Boolean) of object;
property OnBroadCastDecideEvent: TBroadCastOKEvent;
```

Description

This event allows you to code TAstaProviders to only broadcast changes if any fields change. By setting the BroadCastOk argument (default is True) you can decide if you want server side changes to be broadcast or not.

1.8.2.3.2.13 TAstaProvider.OnRefetchSQL

TAstaProvider

Declaration

```
property OnRefetchSQL: TAstaOnRefetchSQLEvent;
type TAstaOnRefetchSQLEvent = procedure (Sender: TObject; ClientSocket:
  TCustomWinSocket; var TheSQL: string; SQLParser: TAstaSQLParser) of
  object;
```

Description

ASTA 3 adds a new event to TAstaProviders: OnRefetchSQL. The AstaSQLParser is passed in so that you can call Deconstruct and be presented with all the Fields, Tables, Tablename and Where Clause. It's up to you to change the TheSQL:string param passed in. You can use the AstaSQLParser and call Construct after you have changed the properties.

Here is an example from the Asta3ADOServer:

```
procedure TAstaDataModule.CustomerProviderRefetchSQL(Sender: TObject;
  ClientSocket: TCustomWinSocket; var TheSQL: string; SQLParser:
  TAstaSQLParser);
begin
  with Sender as TAstaProvider do begin
    TAstaServerSocket(AstaServerSocket).RecordServerActivity(ClientSocke
  t, 'Refetch Provider SQL ' + TheSQL);
  end;
end;
```

Below are some properties of the AstaSQLParser

```
property UpdateTable: string;
property DeleteTable: string;
property InsertTable: string;
property Tables: TStrings;
property Correlations: TStrings;
property Group: TStrings;
property Having: string;
property Order: TStrings;
property Fields: TStrings;
property Where: string;
property SQL: TStrings;
```

1.8.2.3.2.14 TAstaProvider.OnRefetchDataSet

TAstaProvider

Declaration

```
 TAstaOnRefetchDataSetEvent=Procedure (Sender :
TObject;ClientSocket:TCustomWinSocket; TheDataSet:TAstaDataSet;Var
TheDataSetAsString:String) of object;
```

Description

ASTA 3 adds a new event to TAstaProviders: OnRefetchDataSet. This allows for the Dataset, that contains the refetched values to be manipulated and to add other columns and to change values. The DataSet will stream back to the client and be updated using a bookmark.

1.8.2.4 TAstaServerSocket

Properties : Methods : Events

Unit

AstaServer

Declaration

```
type TAstaServerSocket = class(TServerSocket);
```

Description

An AstaServerSocket is the core component of an ASTA server. The ASTA server is the "application server" or "middle tier." It manages all the client connections and simultaneously speaks to the database. All messages in an ASTA application flow through the AstaServerSocket.

See the Developer's Abstract for an overview, and Threading ASTA Servers for a discussion of ASTA threading models.

1.8.2.4.1 Properties

TAstaServerSocket

Active
AddCookie
Address
AstaProtocol
AstaServerName
Compression
DataBaseName
DataBaseSessions
DataSet
DirectoryPublisher
DisconnectClientOnFailedLogin
DisposeofQueriesForThreads
DTAccess
DTPassword

DTUserName
Encryption
Instant Messaging
KeysExchange
LogItems
LogOn
MaximumAsyncSessions
MaximumSessions
MetaDataSet
Port
SecureServer
ServerAdmin
ServerProviderBroadcastList
ServerSQLGenerator
ServerType
SessionList
StatelessListMinutesForExpiration
StatelessListMinutesToCheck
StatelessUserList
StoredProcDataSet
ThreadingModel
TrustedAddresses
UserCheckList
UserList

1.8.2.4.1.1 TAstaServerSocket.Active

TAstaServerSocket

Declaration
```
property Active: Boolean;
```

Description
When the Active property is set to true, an ASTA server starts to listen on the port that it is configured to use.

1.8.2.4.1.2 TAstaServerSocket.AddCookie

TAstaServerSocket

Declaration
```
property AddCookie: TAstaServerCookieOption;
TAstaServerCookieOption = (coNoCookie, coStatelessUserList, coUserCoded);
```

Description
Adds a "cookie" to ASTA messaging packets for stateless HTTP authentication. See Stateless Clients for more detail.

1.8.2.4.1.3 TAstaServerSocket.Address

TAstaServerSocket

Declaration

Description

1.8.2.4.1.4 TAstaServerSocket.AstaProtocol

TAstaServerSocket

Declaration
```
property AstaProtocol: TAstaProtocol;
```

Description
THIS PROPERTY HAS BEEN DEPRECATED IN ASTA 3.

ASTA 3 servers can now handle TCP/IP and HTTP clients with no changes to the server protocol. Each user can have their own protocol. See UserRecord.Protocol.

ASTA servers and clients use a proprietary messaging format with the default transport using TCP/IP. To traverse firewalls ASTA TCP/IP messaging can be made to appear as HTTP messages or actually become HTTP messsages. There is no need to set this property on the AstaServerSocket as ASTA servers can serve TCP/IP and HTTP clients unchanged. The only requirement is that to support stateless HTTP clients, servers must be run using Pooled or Smart Threading.

ASTAClientSockets can call serveral helper methods and run time dialogs to configure ASTA clients to use protocols other than TCP/IP.

1.8.2.4.1.5 TAstaServerSocket.AstaServerName

TAstaServerSocket

Declaration
```
property AstaServerName: string;
```

Description
Use of this property allows a name to be assigned to an ASTA server that can be seen from AstaClientDatasets using the mdDBMSName ordinal of the MetaData property.

1.8.2.4.1.6 TAstaServerSocket.Compression

TAstaServerSocket

Declaration
```
property Compression: TAstaCompression;
```

Description
This property determines if compression will be employed by your application. It works in conjunction with the OnCompress and OnDecompress events.

NOTE: This property must have the SAME SETTING on the ServerSocket AND the ClientSocket.

1.8.2.4.1.7 TAstaServerSocket.DataBaseName

TAstaServerSocket

Declaration

```
property DataBaseName: string;
```

Description
Use of this property allows a DatabaseName to be assigned to an ASTA server that can be seen from AstaClientDatasets using the mdDBMSName ordinal of the MetaData property.

1.8.2.4.1.8 TAstaServerSocket.DataBaseSessions

TAstaServerSocket

Declaration
```
property DatabaseSessions: string;
```

Description
The DatabaseSessions property of the TAstaServerSocket determines the number of sessions or connections that will be pooled by the AstaServer and will be created at startup when using the Pooled Sessions Threading Model and expands on demand. By creating the sessions before the server starts to run, they will be available when needed. You can limit the number of DatabaseSessions that the AstaSessionList explands to by setting the MaxiumumSessions property.

See Threading ASTA Servers for a fuller discussion.

1.8.2.4.1.9 TAstaServerSocket.DataSet

TAstaServerSocket

Declaration
```
property DataSet: TDataset;
```

Description
When running an ASTA server in the Single Thread model, set the the DataSet property to any TDataSet descendent on the server in the OnSubmitSQL event, and that dataset will be streamed back to the ASTA client.

1.8.2.4.1.10 TAstaServerSocket.DirectoryPublisher

TAstaServerSocket

Declaration
```
property DirectoryPublisher: TAstaServerDirectory;
```

Description
See the tutorial RemoteDirectory for more information and an example using this component.

1.8.2.4.1.11 TAstaServerSocket.DisconnectClientOnFailedLogin

TAstaServerSocket

Declaration
```
property DisconnectClientOnfailedLogin: Boolean;
```

Description
When this is true, the client is automatically disconnected if Verified is set to False in the OnClientLogin or OnClientDBLogin events. It is more secure to disconnect a client rather than sending a KillMessage.

Security Issues

1.8.2.4.1.12 TAstaServerSocket.DisposeofQueriesForThreads

TAstaServerSocket

Declaration
```
property DisposeofQueriesForThreads: Boolean;
```

Description
When threading ASTA servers this property determines whether ASTA disposes of a query after a client does some database task. If the ASTA server implements it's queries using a data module (as in the AstaODBCServer example) set DisposeofQueriesForThreads to False.

See Threading ASTA Servers for a fuller discussion.

1.8.2.4.1.13 TAstaServerSocket.DTAccess

TAstaServerSocket

Declaration
```
property DTPassword: string;
```

Description
DTAccess controls which clients have design time access to your AstaServer (and therefore your data).
ASTA provides powerful design time features by allowing a developer to access currently running ASTA servers. This access does not require a password, a convenience that allows you to have easy access to your design time data without having to type in user names and passwords at every turn. It also opens up a potential security hole. Other ASTA developers, if they knew the IP address and the port of your ASTA server, would be able to gain access to your database if they were so inclined. One simple step that you can take to help avoid this problem is to switch your ASTA servers from the default port of 9000 to a different number above 1024.

To close the door completely, ASTA provides server-side and client-side properties that allow you to control who accesses the servers at design time. The AstaClientSocket and the AstaServerSocket both have DTUserName and DTPassword properties. Those properties are controlled by the AstaServerSocket's DTAccess property. DTAccess is defaulted to True. For maximum security, you should set DTAccess to False and then assign matching user name and passwords to the AstaClientSocket and AstaServerSocket DTUserName and DTPassword properties. If you set DTAccess to False, you must provide DTUserName and DTPassword values. Null values will not be accepted and the client will be terminated.

1.8.2.4.1.14 TAstaServerSocket.DTPassword

TAstaServerSocket

Declaration
```
property DTPassword: string;
```

Description
ASTA provides powerful design time features by allowing a developer to access currently

running ASTA servers. This access does not require a password, a convenience that allows you to have easy access to your design time data without having to type in user names and passwords at every turn. It also opens up a potential security hole. Other ASTA developers, if they knew the IP address and the port of your ASTA server, would be able to gain access to your database if they were so inclined. One simple step that you can take to help avoid this problem is to switch your ASTA servers from the default port of 9000 to a different number above 1024.

To close the door completely, ASTA provides server-side and client-side properties that allow you to control who accesses the servers at design time. The AstaClientSocket and the AstaServerSocket both have DTUserName and DTPassword properties. Those properties are controlled by the AstaServerSocket's DTAccess property. DTAccess is defaulted to True. For maximum security, you should set DTAccess to False and then assign matching user name and passwords to the AstaClientSocket and AstaServerSocket DTUserName and DTPassword properties. If you set DTAccess to False, you must provide DTUserName and DTPassword values. Null values will not be accepted and the client will be terminated.

1.8.2.4.1.15 TAstaServerSocket.DTUserName

TAstaServerSocket

Declaration
```
property DTUserName: string;
```

Description
ASTA provides powerful design time features by allowing a developer to access currently running ASTA servers. This access does not require a password, a convenience that allows you to have easy access to your design time data without having to type in user names and passwords at every turn. It also opens up a potential security hole. Other ASTA developers, if they knew the IP address and the port of your ASTA server, would be able to gain access to your database if they were so inclined. One simple step that you can take to help avoid this problem is to switch your ASTA servers from the default port of 9000 to a different number above 1024.

To close the door completely, ASTA provides server-side and client-side properties that allow you to control who accesses the servers at design time. The AstaClientSocket and the AstaServerSocket both have DTUserName and DTPassword properties. Those properties are controlled by the AstaServerSocket's DTAccess property. DTAccess is defaulted to True. For maximum security, you should set DTAccess to False and then assign matching user name and passwords to the AstaClientSocket and AstaServerSocket DTUserName and DTPassword properties. If you set DTAccess to False, you must provide DTUserName and DTPassword values. Null values will not be accepted and the client will be terminated.

1.8.2.4.1.16 TAstaServerSocket.Encryption

TAstaServerSocket

Declaration
```
property Encryption: TAstaEncryption;
```

Description
This property determines the encryption, if any, that will be employed by your application.

Value	Meaning
aeAstaEncrypt	A simple encryption, just set client and server properties.
aeNoEncryption	No encryption is used.
aeUserDefined	Your algorithm is used. Set client and server properties and the OnEncrypt and OnDecrypt event handlers for the client AND server.

NOTE: This property must have the SAME SETTING on the ServerSocket AND the ClientSocket. This property is used in conjunction with the AstaClientSocket's and AstaServerSocket's OnEncrypt and OnDecrypt events.

1.8.2.4.1.17 TAstaServerSocket.InstantMessaging

Declaration

Description

1.8.2.4.1.18 TAstaServerSocket.KeysExchange

Declaration

Description

1.8.2.4.1.19 TAstaServerSocket.LogItems

Declaration

Description

1.8.2.4.1.20 TAstaServerSocket.LogOn

TAstaServerSocket

Declaration
```
property LogOn: Boolean;
```

Description
When the LogOn property is set to true, an ASTA server will show internal debug messages and any messages you have coded on your ASTA server using the

OnShowServerMessage. For production servers it is recommended that NO logging is done as writing to any GUI control consumes quite a bit of CPU resources.

1.8.2.4.1.21 TAstaServerSocket.MaximumAsyncSessions

TAstaServerSocket

Declaration
```
property MaximumAsyncSessions: Integer; default -1;
```

Description
ASTA Messaging can be run threaded and also request Database components within the threads by calling the ThreadedDBUtilityEvent. When running under the Pool Sessions Threading Model ASTA supports Asyncronous Database Access within threads. This an advanced feature that can consume a great deal of resources but can also be very powerful in allowing remote ASTA clients to run concurrent Database processes in threads on ASTA servers.

See Threading ASTA Servers for a fuller discussion.

1.8.2.4.1.22 TAstaServerSocket.MaxiumumSessions

TAstaServerSocket

Declaration
```
property MaximumSessions: Integer; default -1;
```

Description
When running the server in Pooled Sessions threading model at server startup DatabaseSessions number of DataModules are created on the server by calling the ThreadedDBSupplySession event and adding those DataModules the AstaSessionList. The AstaSessionList will expand by demand. You can control the maximum number of Pool Sessions with the MaximumSessions property. When this MaximumSessions property is set and the pool tries to expand beyond that number, Database Requests are Queued until a Session becomes available.

1.8.2.4.1.23 TAstaServerSocket.MetaDataSet

TAstaServerSocket

Declaration
```
property MetaDataSet: TDataset;
```

Description
This is the dataset that is used for meta data requests from ASTA clients. Make sure the OnFetchMetaData event of the TAstaServerSocket sets the MetaData property to the correct dataset after a request from an ASTA client.

1.8.2.4.1.24 TAstaServerSocket.Port

TAstaServerSocket

Declaration
```
property Port: Integer;
```

Description
A socket is a combination of an IP address and port number. You can control the ASTA server port by assigning this property. You can assign port numbers above 2048. The AstaServerSocket defaults to port 9000. If you change to Port 5555, then you will need to make sure that your AstaClientSocket.Port property is assigned 5555.

For more information, see AstaClientSocket.Port

1.8.2.4.1.25 TAstaServerSocket.SecureServer

TAstaServerSocket

Declaration
```
property SecureServer: Boolean;
```

Description
With SecureServer set to true the only message allowed from a client with no UserName already on the UserList, is a LoginRequest or the client gets disconnected. After the loginrequest the UserRecord of the UserList has a UserName so the door allows any remote users with a userName through which means they have already passed the Login. This means all doors are closed except the login door. Design time access is NOT suported with this property set to true.

See Security Issues

1.8.2.4.1.26 TAstaServerSocket.ServerAdmin

TAstaServerSocket

Declaration
```
property ServerAdmin: TServerAdmin;
```

Description
ServerAdmin allows for remote server administration capabilities. See ASTA Server Admin for more information.

1.8.2.4.1.27 TAstaServerSocket.ServerProviderBroadCastList

TAstaServerSocket

Declaration
```
property ServerProviderBroadCastList: TAstaProviderBroadCastList;
```

Description
Internal Datastructure used to handle provider broadcasts. ASTA uses this internally and access at the application level is not normally required.

1.8.2.4.1.28 TAstaServerSocket.ServerSQLGenerator

TAstaServerSocket

Declaration
```
property ServerSQLGenerator: TAstaSQLGenerator;
```

Description

Contains the server side settings to allow for customization of SQL generated by the ASTA server. See TAstaSQLGenerator.

1.8.2.4.1.29 TAstaServerSocket.ServerType

TAstaServerSocket

Declaration
```
type TServerType = (stNonBlocking, stThreadBlocking);
property ServerType: TServerTypeProperty;
```

Description
In ASTA applications, leave the ServerType set to the default, stNonBlocking.

1.8.2.4.1.30 TAstaServerSocket.SessionList

TAstaServerSocket

Declaration
```
property SessionList: TAstaSessionList;
```

Description
For Pooled Sessions, the actual DataModule list used to pool database connections.

1.8.2.4.1.31 TAstaServerSocket.StatelessListMinutesforExpiration

Declaration

Description

1.8.2.4.1.32 TAstaServerSocket.StatelessListMinutesToCheck

Declaration

Description

1.8.2.4.1.33 TAstaServerSocket.StatelessUserList

Declaration

Description

1.8.2.4.1.34 TAstaServerSocket.StoredProcDataSet

TAstaServerSocket

Declaration
```
property StoredProcDataSet: TDataset;
```

Description

Set the StoredProcDataSet to the server side TDataSet descedent that contains a result set after the execution of a stored procedure on the server. Used only in the Single Threaded Model.

1.8.2.4.1.35 TAstaServerSocket.ThreadingModel

TAstaServerSocket

Declaration

```
property ThreadingModel: TAstaThreadModel;
type TAstaThreadModel = (tmSingleSession, tmPooledSessions,
tmPersistentSessions);
```

Description

See Threading ASTA servers for an overview.

Value	Meaning
tmSingleSession	This is the default model and should be the fastest for small numbers of users or larger numbers of users that don't have excessive concurrent activity.
tmPooledSessions	This is a resource conscious scaling model. Use the DataBaseSessions property to set the number of sessions that are created at server startup. All connections are created at server startup, and client requests are paired with sessions from this pool on an as-needed basis. If you are trying to support a large number of users, this is the model to use. The pooled number increases dynamically if there is a demand.
tmPersistentSessions	This allows each client to connect to a session when the client connects to the server. The sessions are persistent. Used in conjunction with the TAstaClientSocket.Login property, you can use this threaded model to allow users to login with their actual database login, and even allow each client to access a separate alias. When the client connects, the socket triggers an event (DBLoginEvent) to create a database session. Any subsequent client database access will pair the incoming socket with the session chosen at startup, within a spawned thread. When the client disconnects, the session is disposed of. When an ASTA server runs in this mode, ASTA clients can fire a server query and keep a cursor open on the server. Instead of fetching all the records in the query the client can request X number of rows via the PacketSize property. See Packet Fetches for a Full discussion of Packetized Querys

1.8.2.4.1.36 TAstaServerSocket.TrustedAddresses

TAstaServerSocket

Declaration
```
property TrustedAddresses: TStrings;
```

Description
Users can be authenticated by ASTA Servers using the OnClientAuthentiate Event and design time access can be granted to certain uses. TrustedAddresses is a more elegant way to secure the server but to allow certain "trusted" addresses access regardless of user name or password. If you set SecureServer to true design time access is blocked UNLESS you include the IP address of any machine you want to be able to access your server. Trusted Addresses basically create a back door for access to the server for any IP address you add to the Trusted Address List.

1.8.2.4.1.37 TAstaServerSocket.UserCheckList

TAstaServerSocket

Declaration
```
property UserCheckList: TCheckListBox;
```

Description
ASTA provides a built-in TUsers object that tracks the users connected to the server. Drop a TCheckListBox component onto the server and then set the UserCheckList to it. When users connect or disconnect the CheckListBox will populate/depopulate automatically.
The UserCheckList will populate with the Socket (the client IP address and port on which it is communicating), the UserName and the ApplicationName (if values are assigned to those properties . . . see AutoLoginDlg).

This screenshot shows a UserCheckList in action.

[NEED SCREENSHOT]

To disconnect specific users, utilize the KillSelectedUsers method. Place an "x" next to the user or users that you wish to disconnect and then issue the KillSelectedUsers command.
To broadcast to specific users, utilize the SendSelectPopup or SendSelectCodedMessage command.

1.8.2.4.1.38 TAstaServerSocket.UserList

Declaration
```
property UserList: TUsers;
```

Description
The AstaServerSocket maintains a UserList that contains a list of TUserRecord objects. It tracks information about connected users. Users are added to the UserList after a successful client login on the AstaServerSocket. There is an event fired when users are added or removed: UserList.OnUserListChange. Each item in the UserList contains a

UserRecord with the UserName (if the user has supplied one for a login), Application Name, Connect time and an FParamList that can be used to store any other "scratch" information that your application may require. The FParamList can be used to store extra information and will include any parameters added to the AstaclientSocket.ClientSocketParams at login time. See the AdvancedLogin Tutorial for code examples of using the ClientSocketParams and accessing the AstaServerSocket.UserRecord.FParamList at login time.

All messaging and database routines on the AstaServerSocket supply access to the ClientSocket making the request. This ClientSocket can be used to look up information from the UserList.

1.8.2.4.2 Methods

TAstaServerSocket

CommandLinePortcheck
DisconnectClientSocket
CreateSessionsForDbThreadsa
EarlyRegisterSocket
KillSelectedUsers
ManualBroadcastProviderChanges
PublishServerDirectories
RecordServerActivity
RegisterClientAutoUpdate
RegisterDataModule
RemoteAddressAndPort
SendBroadcastEvent
SendBroadcastPopUp
SendClientKillMessageSocket
SendCodedMessage
SendCodedParamList
SendCodedStream
SendExceptionToClient
SendSelectCodedMessage
SendSelectPopupMessage
ThreadedDBUtilityEvent
ThreadedUtilityEvent

1.8.2.4.2.1 TAstaServerSocket.AddFileToTransportQueue

TAstaServerSocket

Declaration

```
procedure AddFileToTransportQueue(QName,FileName, LocalFileName: string;
  DeleteFile: Boolean; Chunksize: Integer; ClientSocket: TCustomWinSocket);
```

Description

See the ASTA FTP discussion.

1.8.2.4.2.2 TAstaServerSocket.AddFileToTransportQueueParams

TAstaServerSocket

Declaration
```
procedure AddFileToTransportQueueParams(QName: string; Params:
  TAstaParamList; ClientSocket: TCustomWinSocket);
```

Description
See the ASTA FTP discussion.

1.8.2.4.2.3 TAstaServerSocket.AddUserToStatelessUserList

TAstaServerSocket

Declaration
```
procedure AddUserToStatelessUserList(TheClient: TUserRecord; UserName:
  string);
```

Description
Used internally to add users to the the StatelessUserList.

1.8.2.4.2.4 TAstaServerSocket.BCBThreadedDBUtilityEvent

TAstaServerSocket

Declaration
```
procedure BCBThreadedDBUtilityEvent(ClientSocket: TCustomWinSocket;
  QueryToUse: TThreadDbAction; BCBCustomEvent: BCBThreadUtilityCallBack;
  Params: TAstaParamList);
```

Description
Used for the BCB linker.

1.8.2.4.2.5 TAstaServerSocket.ClientBroadCastParams

TAstaServerSocket

Declaration
```
procedure ClientBroadCastParams(Msgid: Integer; P: TAstaParamList);
  overload;
procedure ClientBroadCastParams(Originator: TCustomWinsocket; Msgid:
  Integer; P: TAstaParamList); overload;
```

Description
Broacasts ParamLists to users. Overloaded so that you can optionally skip the user who triggered a broadcast request (originator).

1.8.2.4.2.6 TAstaServerSocket.ClientUpgradeRegistryInfoAsDataSet

TAstaServerSocket

Declaration
```
function ClientUpgradeRegistryInfoAsDataSet: TAstaDataSet;
```

Description
Returns a TAstaDataSet with auto update registry info. You are responsible for disposing of any DatAset returned as a result of this function call.

1.8.2.4.2.7 TAstaServerSocket.CommandLinePortcheck

TAstaServerSocket

Declaration
```
procedure CommandLinePortcheck;
```

Description
Call this method in an ASTA server before the server sets it's Active property to True if you want to allow the ASTA server command line switches to be checked.

1.8.2.4.2.8 TAstaServerSocket.CreateSessionsForDbThreads

TAstaServerSocket

Declaration
```
procedure CreateSessionsForDBThreads(FreeSessions: Boolean);
```

Description
This is called in an ASTA server when the server first starts and initializes the AstaSessionList when running an ASTA server. If you want ASTA to free the sessions set it to True. The SessionList can be reinitialized at runtime by freeing and re-calling CreateSessionsforDBThreads. This would be necessary if the Database is being backed up and no database connections where allowed.

The following is an example from the AstaDOAServer (Direct Oracle Access) that frees the SessionList and creates it again.

```
procedure TFrmMain.ReLogonToAllSessions;
var
  i: Integer;
  ADM: TastaOracleDataModule;
begin
  try
    if DM.MainLogon.Session.Connected then DM.MainLogon.Session.LogOff;
    DM.MainLogon.Session.LogOn;
    mrequests.lines.add('Main Database reconnected');
  except
    mrequests.lines.add('Problems reconnecting to Main Login' +
  EDataBaseError(ExceptObject).Message);
  end;
  try
    //with a single session there is no SessionList
    if AstaServerSocket1.ThreadingModel=tmSingleSession then Exit;
    for i := 0 to AstaServerSocket1.SessionList.Count - 1 do begin
      //walk the session list and type cast and call whatever routines
  are needed
      //to check to see if it is connected and then re-logon
      Adm := AstaServerSocket1.SessionList.GetSession(i) as
  TAstaOracleDataModule;
```

```
      if Adm.MainLogon.Session.Connected then Adm.MainLogOn.Session.LogOff;
          Adm.MainLogOn.Session.LogOn;
          mrequests.lines.add('Session ' + InttoStr(i) + ' reconnected');
        end;
      except
        mrequests.lines.add('Session List Reconnect Problem.' +
      EDataBaseError(ExceptObject).Message);
        //try again!!!
        AstaServerSocket1.SessionList.Free;
        AstaServerSocket1.SessionList := nil;
        AstaServerSocket1.CreateSessionsForDbThreads(True);
        //this calls the OnDBSupplySession event of the AstaServerSocket
      end;
    end;
```

1.8.2.4.2.9 TAstaServerSocket.DisconnectClientBroadCaster

TAstaServerSocket

Declaration

```
procedure DisconnectClientBroadCaster(ClientSocket: TCustomWinSocket; S:
  string; ClientEncryption: Boolean);
```

Description

Used for ASTA chat tutorials where a list of users is sent to everybody when a client disconnects. Deprecated with the ASTA Instant messaging API.

1.8.2.4.2.10 TAstaServerSocket.DisconnectClientSocket

TAstaServerSocket

Declaration

```
procedure DisconnectClientSocket(ClientSocket: TCustomWinSocket);
```

Description

This method disconnects a remote socket and is more reliable than SendClientKillMessageSocket as that depends on an ASTA client responding to the message from the server.

Security Issues

1.8.2.4.2.11 TAstaServerSocket.EarlyRegisterSocket

TAstaServerSocket

Declaration

```
procedure EarlyRegisterSocket(ClientSocket: TCustomWinSocket);
```

Description

This adds a ClientSocket to the UserList. If you want to communicate with a ClientSocket in the OnClientConnectevent you must add the ClientSocket to theUserList first by calling this method.

1.8.2.4.2.12 TAstaServerSocket.FileSegmentSend

TAstaServerSocket

Declaration
```
procedure FileSegmentSend(ClientSocket: TCustomWinSocket; P:
  TAstaParamList);
```

Description
Used internally to implement FileSegment sends. See the ASTA FTP discussion.

1.8.2.4.2.13 TAstaServerSocket.IsNonVCLClient

TAstaServerSocket

Declaration
```
function IsNonVCLClient(ClientSocket: TCustomWinSocket): Boolean;
```

Description
Returns true if the Client is a non-vcl or SkyWire client. See ASTA SkyWire.

1.8.2.4.2.14 TAstaServerSocket.KillSelectedUsers

TAstaServerSocket

Declaration
```
procedure KillSelectedUsers(KillMessage: string);
```

Description
The KillSelectedUsers method is used in conjunction with the UserCheckList property. The UserCheckList will present a visual list that will allow you to select one or more users. Once the users have been selected, issuing the KillSelectedUsers command will terminate the client connections to the server.

```
procedure TForm1.Button5Click(Sender: TObject);
begin
  AstaServerSocket1.KillSelectedUsers('Your application is being ' +
    'terminated by the System Administrator.');
end;
```

1.8.2.4.2.15 TAstaServerSocket.LogServerActivity

TAstaServerSocket

Declaration
```
procedure LogServerActivity(LogItems: TAstaLogItems; S: TCustomWinSocket;
  const Msg: string);
```

Description
ASTA servers, by default come with logging turned on but with an option to turn it off. By definition, logging is an expensive activity and is not thread safe. Writing to a UI component (TMemo) is not thread safe and eats a lot of CPU cycles.

ASTA supports logging by coding the AstaServerSocket OnShowServerMessage event. Internally the RecordServerActivity Method calls OnShowServerMessage. There is a Logon:Boolean property that allows for all logging to be turned on or off and ASTA 3 servers allow this to be controlled by the Server UI or with the RemoteAdmin feature.

If you require more granualar control of the logging process, you can define which events you want the server to log.

```
TAstaLogItem = (alLogAll, alThread, alException, alProvider,
  alServerMethod, alProviderBroadCast, alRefetch, lLogin,
  alClientUpdate, alPersistentSession, alJava, alGeneral, alDisconnect,
  alHttp, alUtilityThread, alCreateSession, alDisposeOfSession,
  alSendExceptionToClient, alThreadError, alSelectSQL, alExecSQL,
  alMetaData, alStoredProcs, lFailedLogins, alPdas, alSessionCheckOut);

TAstaLogItems = set of TAstaLogItem;
TServerLogEvent = procedure (Sender: TObject; LogItems: TAstaLogItems;
  ClientSocket: TCustomWinSocket; Msg: string; var RecordOnMainLog:
  Boolean) of object;

property NoLogItems: TAstaLogItems;
property LogItems: TAstaLogItems;
property LogOn: Boolean;
```

You can specify that you want to log all items and add items that you don't want to log to the NoLogItems property, or you can define the actual items you want logged.

```
property LogEvent: TServerLogEvent;
TServerLogEvent = procedure (Sender: TObject; LogItems: TAstaLogItems;
  ClientSocket: TCustomWinSocket; Msg: string; var RecordOnMainLog:
  Boolean) of object;
```

Internally ASTA calls:

```
procedure LogServerActivity(LogItems: TAstaLogItems; S:
  TcustomWinSocket; const Msg: string);
```

Which checks the LoginOn, LogItems and NoLogItems. If the LogEvent is coded you can further control if an item is passed through to the log. Only if all the tests passed is RecordServerActivity actually called.

Note: Problems on ASTA servers are often caused by ASTA users writing logging routines that are not thread safe. Beware of too much logging!

1.8.2.4.2.16 TAstaServerSocket.ManualBroadCastProviderchanges

TAstaServerSocket

Declaration

```
procedure ManualBroadCastProviderChanges(const ProviderName,
  UpdateTableName: string; D: TAstaDataSet); overload;
procedure ManualBroadCastProviderchanges(Provider: TAstaProvider);
  overload;
```

Description

1.8.2.4.2.17 TAstaServerSocket.NetWorkProviderBroadcast

TAstaServerSocket

Declaration
```
procedure NetWorkProviderBroadcast(TableName: string);
```

Description
By calling this method, all TAstaProviders that use the same UpdateTableName will broadcast to any registered users. This is useful if you have a provider that returns different fields than other providers yet you want broadcasted changes to go to the same clients.

1.8.2.4.2.18 TAstaServerSocket.PdaFastEnable

TAstaServerSocket

Declaration
```
procedure PdaFastEnable(CodedParamListIntercept: Boolean);
```

Description
Enables support for Palm and WinCE using the SkyWire API. See ASTA SkyWire discussion.

1.8.2.4.2.19 TAstaServerSocket.PublishedDirectoriesFromRegistry

TAstaServerSocket

Declaration
```
function PublishedDirectoriesFromRegistry: TAstaDataSet;
```

Description
Returns an in memory DataSet that represents the "Aliases" used in the Remote Directory Components. This DataSet can be edited and then any changes can be updated to the registry by calling UpdatePublishedDirectoriesToRegistry, This call creates a TAstaDataSet which you are responsible for disposing of.

1.8.2.4.2.20 TAstaServerSocket.PublishServerDirectories

TAstaServerSocket

Declaration
```
procedure PublishServerDirectories;
```

Description
Allows server side directories to be "aliased" and made available for ASTA clients to view and send files back and forth from using the TAstaRemoteDirectory component.

See Remote Directory Components for a full discussion.

1.8.2.4.2.21 TAstaServerSocket.RecordServerActivity

TAstaServerSocket

Declaration
```
procedure RecordServerActivity(S: TCustomWinSocket; Msg: string);
```

Description
If you have coded the OnShowServerMessage event of the TAstaServerSocket, then RecordServerActivity will call that event. This is used internally by ASTA to show threading information and other database activity.

1.8.2.4.2.22 TAstaServerSocket.RegisterClientAutoUpdate

TAstaServerSocket

Declaration
```
procedure RegisterClientAutoUpdate;
```

Description
In order to take advantage of ASTA's built-in automatic client update capabilities, you must register the new client executable at the server. RegisterClientAutoUpdate brings up a dialog that will allow you to browse for any EXE's that you want your remote clients to have available. There is also a remote interface available to allow you to see all available server EXE's and allow you to update the list (which is stored in the registry), and even to send EXE's to the server to be placed in the correct directory. Se also AstaClientSocket.RequestUtilityInfo for more details

```
procedure TForm1.RegisterUpdateClient1Click(Sender: TObject);
begin
  AstaServerSocket1.RegisterClientAutoUpdate;
end;
```

See Automatic Client Updates for a full discussion.

1.8.2.4.2.23 TAstaServerSocket.RegisterDataModule

TAstaServerSocket

Declaration
```
procedure RegisterDataModule(DM: TComponent)
```

Description
This is called at the startup of an ASTA server and all of the TDataSet descendents, TAstaProviders and TAstaBusinessObjectManagers are registered so that they can be seen by ASTA clients.

Example

```
AstaServerSocket1.RegisterDataModule(Self);
```

1.8.2.4.2.24 TAstaServerSocket.RemoteAddressAndPort

TAstaServerSocket

Declaration
```
function TAstaServerSocket.RemoteAddressAndPort(Socket: TCustomWinSocket):
  string;
```

Descripiton
Provides a quick and easy way to return a string representing a ClientSocket's remote

address and port number.

1.8.2.4.2.25 TAstaServerSocket.SendAsyncException

TAstaServerSocket

Declaration
```
procedure SendAsyncException(ClientSocket: TCustomWinSocket; ErrorToken:
  Integer; const Msg: array of string);
```

Description
Used to raise an exception on the client when connected asyncronously.

1.8.2.4.2.26 TAstaServerSocket.SendAsyncExceptionUserRecord

TAstaServerSocket

Declaration
```
procedure SendAsyncExceptionUserRecord(const TheClient: TUserRecord;
  ErrorToken:Integer;Const Msg: array of string); overload;
procedure SendAsyncExceptionUserRecord(const TheClient: TUserRecord;
  ErrorToken: Integer; TheParamList: TAstaParamList); overload;
```

Description
Used to raise an exception on the client when connected asyncronously.

1.8.2.4.2.27 TAstaServerSocket.SendBroadcastEvent

TAstaServerSocket

Declaration
```
procedure SendBroadcastEvent(S: string);
```

Description
There are two methods that allow you to send broadcasts to connected clients.
The fist method, SendBroadcastPopup is intrusive and will display the message in a ShowMessage dialog box. This type of broadcast is appropriate for administrative messages.
The second method, SendBroadcastEvent, allows you to broadcast a message and control the way that it appears at the client. It must be used in conjunction with the AstaClientSocket's OnServerBroadcast event.

The following code shows how the AstaClientSocket's OnServerBroadcast event would copy a SendBroadcastEvent method to a memo.

```
procedure TForm1.AstaClientSocket1ServerBroadcast(Sender: TObject;
  S: string);
begin
  mServerBroadcasts.Lines.Add(S);
  mServerBroadcasts.Lines.Add(Chr(VK_Return));
end;
```

1.8.2.4.2.28 TAstaServerSocket.SendBroadcastPopUp

TAstaServerSocket

Declaration
```
procedure SendBroadcastPopUp(S: string);
```

Description
There are two methods that allow you to send broadcasts to connected clients. The fist method, SendBroadcastPopup is intrusive and will display the message in a ShowMessage dialog box. This type of broadcast is appropriate for administrative messages.

The second method, SendBroadcastEvent, allows you to broadcast a message and control the way that it appears at the client. It must be used in conjunction with the AstaClientSocket's OnServerBroadcast event.

1.8.2.4.2.29 TAstaServerSocket.SendClientKillMessage

TAstaServerSocket

Declaration
```
procedure SendClientKillMessage(TheClient: TUserRecord; KillMessage:
string);
```

Description
SendClientKillMessage will disconnect the remote client after showing the KillMessage. The client application will be terminated unless the AstaClientSocket.OnTerminateMessageFromServer is coded. For extended security use DisconnectClientSocket so as not to allow another ASTA user to code a client application that ignores kill messages.

Security Issues

1.8.2.4.2.30 TAstaServerSocket.SendClientKillMessageSocket

TAstaServerSocket

Declaration
```
procedure SendClientKillMessageSocket(Socket: TCustomWinSocket;
KillMessage: string);
```

Description
SendClientKillMessageSocket will disconnect the remote socket after showing the KillMessage.

1.8.2.4.2.31 TAstaServerSocket.SendClientPopupMessage

Declaration

Description

1.8.2.4.2.32 TAstaServerSocket.SendCodedMessage

TAstaServerSocket

Declaration
```
procedure SendCodedMessage(ClientSocket: TCustomWinsocket; MsgID: Integer;
Msg: string);
```

Description
SendCodedMessage allows you to send special messages throughout an ASTA application. The SendCodedMessage method allows you to roll your own protocol where certain MsgIDs have specific meaning to your application. The ability to create your own protocol in this fashion is a simple yet powerful technique. See the Simple Business Objects code below.

The simple code below sends a message from a client to the server, but the message is sent with two different MsgIDs. The server will handle the message differently depending on which of the MsgIDs, 1700 or 1750, accompanies the message.

```
procedure TForm1.Button4Click(Sender: TObject);
begin
  AstaClientSocket1.SendCodedMessage(1700, eClientCodedMessage.Text);
end;

procedure TForm1.Button3Click(Sender: TObject);
begin
  AstaClientSocket1.SendCodedMessage(1750, eClientCodedMessage.Text);
end;
```

The server responds wtih the following implementation code.

```
procedure TForm1.AstaServerSocket1CodedMessage(Sender: TObject;
  ClientSocket: TCustomWinSocket; MsgID: Integer; Msg: string);
begin
  case MsgID of
    1700: AstaServerSocket1.SendCodedMessage(ClientSocket, MsgID,
  UpperCase(Msg));
    1750: AstaServerSocket1.SendCodedMessage(ClientSocket, MsgID,
  LowerCase(Msg));
  end;
end;
```

When returned to the client, it is picked up in the AstaClientSocket's OnCodedMessage event.

Simple Business Objects
The following code shows how you could send CodedMessages to a business object instantiated on the server. In this scenario, 1800 and 1850 are "protocol codes" that you control. The client can call those codes, with or without a parameter and invoke a custom response from the server.

```
procedure Form1.AstaServerSocket1CodedMessage(Sender: TObject;
  ClientSocket: TCustomWinSocket; MsgID: Integer; Msg: string);
begin
  case MsgID of
    1800: begin
            if MyBusinessObject.ChargeSalesTax(StrToInt(Msg)) then
              AstaServerSocket1.SendCodedMessage(ClientSocket, MsgID,
  'TRUE')
            else
              AstaServerSocket1.SendCodedMessage(ClentSocket, MsgID,
  'FALSE');
          end;
    1850: begin
```

```
        AstaServerSocket1.SendCodedMessage(ClientSocket, MsgID,
                  MyBusinessObject.NewCreditLimit(Msg));
              end;
        end;
      end;
```

1.8.2.4.2.33 TAstaServerSocket.SendCodedMessageSocket

TAstaServerSocket

Declaration

```
procedure SendCodedMessageSocket(const TheClient: TUserRecord; MsgID:
  integer; const Msg: string);
```

Description

Same as SendCodedMessage but takes a TUserRecord rather than a ClientSocket: TCustomWinSocket.

1.8.2.4.2.34 TAstaServerSocket.SendCodedParamList

TAstaServerSocket

Declaration

```
procedure SendCodedParamList(ClientSocket: TCustomWinSocket; MsgID:
  Integer; Params: TAstaParamList);
```

Description

SendCodedParamList allows you to send special messages throughout an ASTA application. The SendCodedParamList method allows you to roll your own protocol where certain MsgIDs have specific meaning to your application. The ability to create your own protocol in this fashion is a simple yet powerful technique. See AstaClientSocket.OnCodedParamlist on how to code this procedure on the client. Normally messaging calls are not threaded but ASTA Servers by default will thread messages that use SendGetCodedParamList. If you want to thread messages sent from the server then you must use the AstaServerSocket.ThreadedUtilityEvent.

1.8.2.4.2.35 TAstaServerSocket.SendCodedParamListSocket

TAstaServerSocket

Declaration

```
procedure SendCodedParamListSocket(TheClient: TUserRecord; MsgID: Integer;
Params: TAstaParamList);
```

Description

SendCodedParamListSocket works just like SendCodedParamList except it passes in a full copy of the UserRecord which the TAstaThread copies when it is created. This allows a thread safe use of the UserRecord since it is a copy and not the actual UserRecord of the AstaServerSocket.UserList.

1.8.2.4.2.36 TAstaServerSocket.SendCodedStreamSocket

TAstaServerSocket

Declaration

```
procedure SendCodedStreamSocket(const TheClient: TUserRecord; MsgID:
  integer; MS: TMemoryStream);
```

Description

Same as SendCodedStream but takes a UserRecord rather than a ClientSocket: TCustomWinSocket.

1.8.2.4.2.37 TAstaServerSocket.SendCodedStream

TAstaServerSocket

Declaration

```
procedure SendCodedStream(ClientSocket: TCustomWinSocket; MsgID: Integer;
MS: TMemoryStream);
```

Description

The SendCodedStream compliments the SendCodedMessage method. It can be used to send streams throughout an ASTA application. The client server and then the server can respond with a custom action according to the MsgID. The following code demonstrates the usage.

The client sends a message to the server.

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  AstaClientSocket1.SendCodedMessage(2000, '');
end;
```

The server responds by loading a memo into a stream and sending it to the client.

```
procedure TForm1.AstaServerSocket1CodedMessage(Sender: TObject;
  ClientSocket: CustomWinSocket; MsgID: Integer; Msg: string);
var
  MS : TMemoryStream;
begin
  case MsgID of
    2000: begin
            MS := TMemoryStream.Create;
            mSelectSend.Lines.SaveToStream(MS);
            AstaServerSocket1.SendCodedStream(ClientSocket, 2000, MS);
            MS.Free;
          end;
  end;
end;
```

The client receives the stream and displays it in a memo and saves it to a file.

```
procedure TForm1.AstaClientSocket1CodedStream(Sender: TObject; MsgID:
  Integer; MS: TMemoryStream);
begin
  case MsgID of
```

```
      2000: begin
              mFileFromServer.Lines.Clear;
              mFileFromServer.Lines.LoadFromStream(MS);
              mFileFromServer.Lines.SaveToFile('ATGTestFile.Txt');
            end;
    end;
  end;
```

1.8.2.4.2.38 TAstaServerSocket.SendExceptionToClient

TAstaServerSocket

Declaration

```
procedure SendExceptionToClient(TheSocket: TcustomWinSocket; ErrorList:
  TAstaParamList); overload;
procedure SendExceptionToClient(TheSocket: TcustomWinSocket; ErrorMsg:
  string); overload;
```

Descripiton

This sends the ErrorMsg to an ASTA client and raises an exception on the client.

1.8.2.4.2.39 TAstaServerSocket.SendInstantMessage

TAstaServerSocket

Declaration

```
function SendInstantMessage(Origin: TUserRecord; UserName: string; MsgID:
  integer; Params: TAstaParamList): Boolean;
```

Description

Used internally by the Asta IMA (Instant Message API) to sends an Instant Message to a specific user.

1.8.2.4.2.40 TAstaServerSocket.SendSelectCodedMessage

TAstaServerSocket

Declaration

```
procedure SendSelectCodedMessage(MsgID: Integer; S: string);
```

Description

This property is used in conjunction with the UserCheckList property. You can place an "x" next to one or more users and server messages will be selectively broadcast to the selected users. Coded messages allow you to create your own protocol (see SendCodedMessage) for details.

Example

```
procedure TForm1.Button6Click(Sender: TObject);
begin
  AstaServerSocket1.SendSelectCodedMessage(500, 'This message ' +
    'will be sent to selected individuals, and handled ' +
    'by the OnCodedMessage event handler at the client');
end;
```

See also: SendSelectPopupMessage.

1.8.2.4.2.41 TAstaServerSocket.AstaThreadCount

TAstaServerSocket

Declaration
```
function AstaThreadCount: Integer;
```

Description
Returns the current number of active threads.

1.8.2.4.2.42 TAstaServerSocket.SendSelectPopupMessage

TAstaServerSocket

Declaration
```
procedure SendSelectPopupMessage(S: string);
```

Description
This property is used in conjunction with the UserCheckList property. You can place an "x" next to one or more users and server messages will be selectively broadcast to the selected users.

Example

```
procedure TForm1.Button6Click(Sender: TObject);
begin
  AstaServerSocket1.SendSelectPopupMessage('This message only goes ' +
    'to selected users');
end;
```

See also: SendSelectCodedMessage.

1.8.2.4.2.43 TAstaServerSocket.SendUserNameCodedParamList

TAstaServerSocket

Declaration
```
function SendUserNameCodedParamList(Origin: TUserRecord; UserName: string;
  MsgID: integer; Params: TAstaParamList): Boolean;
```

Description
Used Internally by the Asta IMA (Instant Message API) to sends an Instant Message to a specific user by UserName.

1.8.2.4.2.44 TAstaServerSocket.SetAESKey

TAstaServerSocket

Declaration
```
procedure SetAESKey(AKey: PAESKey128; KeyType: AesKeyType);
```

Description
Low level call to set the AES Keys. SetAESKeyStrings is recommended.

1.8.2.4.2.45 TAstaServerSocket.SetDESStringKey

TAstaServerSocket

Declaration
```
procedure SetDESStringKey(const AKey: string);
```

Description
Sets the key if you are using DES Encryption.

1.8.2.4.2.46 TAstaServerSocket.SessionCount

TAstaServerSocket

Declaration
```
function SessionCount: Integer;
```

Description
Returns the number of Database Sessions if running Pooled or Smart Threading.

1.8.2.4.2.47 TAstaServerSocket.SetAESKeyStrings

TAstaServerSocket

Declaration
```
procedure SetAESKeyStrings(const InKey, OutKey: string);
```

Description
Sets AES Encryption Keys.

1.8.2.4.2.48 TAstaServerSocket.SetDESKey

TAstaServerSocket

Declaration
```
procedure SetDESKey(AKey: PDESKey);
```

Description
Low level call to Set the DES key. SetDESStringKey is recommended.

1.8.2.4.2.49 TAstaServerSocket.StatelessClient

TAstaServerSocket

Declaration
```
function StatelessClient(Clientsocket: TCustomWinSocket): Boolean;
  overload;
function StatelessClient(U: TUserRecord): Boolean; overload;
```

Description
Returns True if the client is running stateless. This could be an HTTP client, a blocking client or a SkyWire Client.

1.8.2.4.2.50 TAstaServerSocket.ThreadedDBUtilityEvent

TAstaServerSocket

Declaration

```
procedure ThreadedDBUtilityEvent(ClientSocket: TCustomWinSocket;
  QueryToUse: TThreadDBAction; CustomEvent: TAstaDBUtilityEvent; Params:
  TAstaParamList);

type TAstaDBUtilityEvent = procedure(Sender: TObject;
  ServerSocket:TAstaServerSocket; ClientSocket: TCustomWinSocket; AQuery:
  TComponent; Params: TAstaParamList) of object;
```

Description

Asta servers are non-blocking and event driven so normal messaging calls are not threaded. You can have ASTA launch a thread in response to a message from any ASTA client and ASTA will also make available some database components for you to use in the thread. If you do not need access to any database calls use ThreadedUtilityEvent. Set the QueryToUse parameter to the type of query component you need like ttSelect or ttExec.

The following shows the use of the ThreadedDBUtilityEvent event from an ASTA server. Notice the AstaServerSocket SendCodedParamListSocket call which uses the ThreadedClient copy of the UserRecord. Note: If a client is disconnected at this point the thread will be marked as terminated, so perhaps a call to if not TAstaThread(Sender).HasTerminted then Exit is needed before the actual send is made.

This example assumes that the TestUtilSQLEvent is being called from within OnCodedParamList on the server which gets called with a ClientSocket:TCustomWinSocket and an incoming Params:TAstaParamList. In this example a request is made for a component that can handle SQLSelects by passing in ttSelect. ThreadedDBSupplyQuery is called with ttSelect to return the Query:TComponent that can be used safely in the thread. The TestUtilSQLEvent must be of type TAstaDBUtilityEvent.

The call is initiated by calling `ThreadedDBUtilityEvent(ClientSocket, ttSelect, TestUtilSQLEvent, Params);`

```
procedure TForm1.TestUtilSQLEvent(Sender: TObject; ServerSocket:
  TAstaServerSocket; ClientSocket: TCustomWinSocket; Query: TComponent;
  Params: TAstaParamList);
var
  p: TAstaParamList;
  s: string;
begin
  //the Sender is now a TAstaThread
  try
    TQuery(Query).Close;
    TQuery(Query).SQL.Text := Params[0].AsString;
    TQuery(Query).Open;
  except
    ServerSocket.SendExceptionToClient(ClientSocket,
      Exception(exceptObject).Message);
    Exit;
  end;
  p := TAstaParamList.Create;
  p.FastAdd(IntToStr(TQuery(Query).RecordCount) + ' records
```

```
    retrieved.');
    p.FastAdd(Tquery(Query).SQL.Text);
    p.FastAdd(CloneDataSetToString(Query as TQuery));
    //don't call the AstaServerSocket1 as we are in a thread
    //use the passed in ServerSocket
    ServerSocket.SendCodedParamListSocket(TAstaThread(Sender).ThreadedClie
    nt, 1550, P);
    //Asta disposes of the query
    //if you want additional components you can call
    //ThreadedDBSupplyQuery since you now have the Sender TAstaThread
    //and the client socket
    p.Free;
  end;
```

See the AstaBDEServer for an example of how to use this method.

1.8.2.4.2.51 TAstaServerSocket.ThreadedUtilityEvent

TAstaServerSocket

Declaration

```
procedure ThreadedUtilityEvent(ServerSocket: TAstaServerSocket;
  ClientSocket: TCustomWinSocket; CustomEvent: TAstaUtilityEvent; P:
  TAstaParamList);
```

Description

Asta servers are non-blocking and event driven so normal messaging calls are not threaded. You can have ASTA launch a thread in response to a message from any ASTA client. If you want to use some database calls in this event use the ThreadedDBUtilityEvent.

See the AstaBDEServer for an example of how to use this method.

The ServerSocket is passed in so that you do not have to access the ServerSocket directly across Thread boundries. The ClientSocket is the actual Socket that is available to each client connected to the server. The CustomEvent your own custom procedure submitted for
threading. It must be a threaded procedure and should take the following format of its declaration which is declared as TAstaUtilityEvent.

```
  procedure NameOfThreadedProcedure(Sender: TObject; ServerSocket:
    TAstaServerSocket; ClientSocket: TCustomWinSocket; Params:
    TAstaParamList);
```

And in the implementation, a sample would be:

```
  procedure TForm1.NameOfThreadedProcedure(Sender: TObject; ServerSocket:
    TAstaServerSocket; ClientSocket: TCustomWinSocket; Params:
    TAstaParamList);
  // var your variables here;
  begin
    // your thread-safe codes here
  end;
```

1.8.2.4.2.52 TAstaServerSocket.UseThreadsSocket

TAstaServerSocket

Declaration
```
function UseThreadsSocket(ClientSocket: TCustomWinSocket):
  TAstaThreadModel;
```

Description
Returns the TAstaThreading Model for any client. Used in tmSmartThreading where users use Pooled Sessions by default but any single user can run a persistent session and switch back and forth from pooled to persistent.

1.8.2.4.2.53 TAstaServerSocket.UseThreads

TAstaServerSocket

Declaration
```
function UseThreads(TheClient: TUserRecord = nil): TAstaThreadModel;
```

Description
Returns the TAstaThreading Model for any client. Used in tmSmartThreading where users use Pooled Sessions by default but any single user can run a persistent session and switch back and forth from pooled to persistent.

1.8.2.4.2.54 TAstaServerSocket.UpdatePublishedDirectoriesToRegistry

TAstaServerSocket

Declaration
```
procedure UpdatePublishedDirectoriesToRegistry(D: TAstaDataSet);
```

Description
Updated the registry replacing the contents that map to "Aliases" used in the Remote Directory Components. The TAstaDataSet is fired obtained by calling PubliishedRemoteDirectoriesFromRegistry.

1.8.2.4.2.55 TAstaServerSocket.User

TAstaServerSocket

Declaration
```
function User(ClientSocket: TCustomWinSocket): TUserRecord; overload;
function User(UserName: string): TUserRecord; overload;
```

Description
Returns the TUsersRecord by using the ClientSocket (TCustomWinSocket) or UserName.

Note: ASTA 3 uses the ClientSocket.Data (Pointer) to store a pointer to the UserRecord so DO NOT use this property.

1.8.2.4.2.56 TAstaServerSocket.UserRecordNil

TAstaServerSocket

Declaration
```
function TAstaServerSocket.UserRecordNil(const TheClient: TUserRecord):
  Boolean;
```

Description
Returns True if TheClient is NIL.

1.8.2.4.2.57 TAstaServerSocket.UtilityInfo

TAstaServerSocket

Declaration

Description

1.8.2.4.2.58 TAstaServerSocket.ValidSocket

TAstaServerSocket

Declaration
```
function ValidSocket(ClientSocket: TCustomWinSocket): Boolean;
```

Description
Returns true if the ClientSocket is not nil and is valid using the same low level Winsock call as VerifyClientConnection. ValidSocket will not force a disconnect if invalid but VerifyClientConnection will.

1.8.2.4.2.59 TAstaServerSocket.VerifyClientConnection

TAstaServerSocket

Declaration
```
procedure VerifyClientConnection(S: TCustomWinSocket);
```

Description
Called internally by the TAstaServerSocket whenever a TCP/IP error is raised, this call uses a low level Winsock API call to verify the client and if not valid force a disconnect.

1.8.2.4.2.60 TAstaServerSocket.VerifyClientConnections

TAstaServerSocket

Declaration
```
procedure VerifyClientConnections;
```

Description
Calls VerifyClientConnection for every connected client in the AstaServerSocket.UserList.

1.8.2.4.3 Events

TAstaServerSocket
LogEvent
OnAccept
OnAddDataModule
OnAstaClientUpdate
OnAstaClientUpdateDecide
OnBeforeProcessToken
OnBeforeServerMethod
OnChatLine
OnCheckOutSession
OnClientAuthenticate
OnClientConnect
OnClientDBLogin
OnClientExceptionMessage
OnClientLogin
OnClientValidate
OnCodedMessage
OnCodedParamList
OnCodedStream
OnCompress
OnCustomAutoIncFormat
OnDecompress
OnDecrypt
OnEncrypt
OnExecRowsAffected
OnExecSQLParamList
OnFetchBlobEvent
OnFetchMetaData
OnInsertAutoIncrementFetch
OnInstantMessage
OnOutCodedDBParamList
OnOutCodedParamList
OnProviderConvertParams
OnProviderSetParams
OnServerQueryGetParams
OnServerQuerySetParams
OnShowServerMessage
OnStatelessClientCheckIn
OnStatelessClientCookie
OnStatelessPacketEvent
OnStoredProcedure
OnSubmitSQL
OnSubmitSQLParams
OnTransactionBegin
OnTransactionEnd
OnTrigger
OnTriggerDefaultValue
OnUserEncryption
OnUserListChange
OnUserRecordStateChange
PersistentFieldTranslateEvent

ThreadedDBSupplyQuery
ThreadedDBSupplySession

1.8.2.4.3.1 TAstaServerSocket.LogEvent

TAstaServerSocket

Declaration
```
procedure LogEvent(ClientSocket: TCustomWinSocket; const Msg: string);
```

Description
ASTA servers, by default come with logging turned on but with an option to turn it off. By definition, logging is an expensive activity and is not thread safe. Writing to a UI component (TMemo) is not thread safe and eats a lot of CPU cycles.

ASTA supports logging by coding the AstaServerSocket OnShowServerMessage event. Internally the RecordServerActivity Method calls OnShowServerMessage. There is a Logon:Boolean property that allows for all logging to be turned on or off and ASTA 3 servers allow this to be controlled by the Server UI or with the RemoteAdmin feature.

If you require more granualar control of the logging process, you can define which events you want the server to log.

```
TAstaLogItem = (alLogAll, alThread, alException, alProvider,
  alServerMethod, alProviderBroadCast, alRefetch, lLogin,
  alClientUpdate, alPersistentSession, alJava, alGeneral, alDisconnect,
  alHttp, alUtilityThread, alCreateSession, alDisposeOfSession,
  alSendExceptionToClient, alThreadError, alSelectSQL, alExecSQL,
  alMetaData, alStoredProcs, lFailedLogins, alPdas, alSessionCheckOut);

TAstaLogItems = set of TAstaLogItem;
TServerLogEvent = procedure (Sender: TObject; LogItems: TAstaLogItems;
  ClientSocket: TCustomWinSocket; Msg: string; var RecordOnMainLog:
  Boolean) of object;

property NoLogItems: TAstaLogItems;
property LogItems: TAstaLogItems;
property LogOn: Boolean;
```

You can specify that you want to log all items and add items that you don't want to log to the NoLogItems property, or you can define the actual items you want logged.

```
property LogEvent: TServerLogEvent;
TServerLogEvent = procedure (Sender: TObject; LogItems: TAstaLogItems;
  ClientSocket: TCustomWinSocket; Msg: string; var RecordOnMainLog:
  Boolean) of object;
```

Internally ASTA calls:

```
procedure LogServerActivity(LogItems: TAstaLogItems; S:
  TcustomWinSocket; const Msg: string);
```

Which checks the LoginOn, LogItems and NoLogItems. If the LogEvent is coded you can further control if an item is passed through to the log. Only if all the tests passed is RecordServerActivity actually called.

Note: Problems on ASTA servers are often caused by ASTA users writing logging routines that are not thread safe. Beware of too much logging!

1.8.2.4.3.2 TAstaServerSocket.OnAccept

TAstaServerSocket

Declaration
```
property OnAccept: TSocketNotifyEvent;
```

Description
Occurs on server sockets just after the connection to a client socket is accepted. For more information see the TServerSocket.OnAccept in the Delphi help file.

1.8.2.4.3.3 TAstaServerSocket.OnAddDataModule

TAstaServerSocket

Declaration
```
property OnAddDataModule: TAddDataModuleEvent;
```

Description
This allows you to add multiple data modules to ASTA servers for support of ASTA server side components and to perform any intialization for the additional data module. Traditionally, a main data module will contain a TDataBase type of component that will need to be used by the additional data module.
In order for ASTA to properly thread ASTA Servers, data modules need to be created by ASTA on demand and OnAddDataModule allows this to occur.

Example

```
procedure TForm1.AstaServerSocket1AddDataModule(Sender: TObject;
  Session: TComponent; DMList: TList);
var
  dm: TMyDataModule;
  i: Integer;
begin
  dm := TMyDataModule.Create(nil);
  for i := 0 to dm.ComponentCount - 1 do
    if dm.Components[i] is TQuery then
      TQuery(Components[i]).DatabaseName :=
        TMyMainDataModule(Sesssion).DatabaseName;
  DMList.Add(dm);
end;
```

1.8.2.4.3.4 TAstaServerSocket.OnAstaClientUpdate

TAstaServerSocket

Declaration
```
property OnAstaClientUpdate:  TAstaAutoClientUpateEvent;
type TAstaAutoClientUpdateEvent = procedure(Sender: TObject;
  UserName, AppName, OldVer, NewVer: string) of object;
```

Description

The OnAstaClienUpdate event allows you to keep a server log or database record of which users have been updated.To control which clients are allowed to get new versions see the OnAstaClientUpdateDecide event. To be able to manage different versions of client exe's that are available for udpates on the server, see Automatic Client Updates.

```
procedure TForm1.AstaServerSocket1AstaClientUpdate(Sender: TObject;
  UserName, AppName, OldVer, NewVer: string);
begin
  // write to your log or database here
end;
```

1.8.2.4.3.5 TAstaServerSocket.OnAstaClientUpdateDecide

TAstaServerSocket

Declaration

```
property OnAstaClientUpateDecide: TAstaClientUpgradeDecideEvent;
type TAstaClientUpgradeDecideEvent = procedure(Sender: TObject;
  ClientSocket: TCustomWinSocket; AppName, OldVer,
  UserName: string; var AllowUpgrade: Boolean) of object;
```

Description

The OnAstaClientUpdateDecide event allows you to control whether remote clients have a new client exe streamed down to them. The ApplicationName, OldVer and UserName arguments are passed to the event and it's up to you to set the AllowUpgrade argument to False. The default is True to allow for upgrades.

1.8.2.4.3.6 TAstaServerSocket.OnBeforeProcessToken

TAstaServerSocket

Declaration

```
property OnBeforeProcessToken: TAstaBeforeProcessTokenEvent;
type TAstaBeforeProcessTokenEvent = procedure(Sender: TObject; TheClient:
  UserRecord; Token: TAstaToken; var Allow: Boolean) of object;
```

Description

This event is fired before any token is processed. A token is the type of message that is being received from a client.The primary use of this is to make some sort of restriction on what operations can be performed by certain user names etc. If you set allow to False, an exception will be raised on the client.

This is a low level call is not only used to heavily customize a server. Typically this event is not required to be coded.

As an example, take a user "TestUser" and you want to restrict it to only run server methods:

```
procedure TfrmMain.AstaServerBeforeProcessToken(Sender: TObject;
  TheClient: UserRecord; Token: TAstaToken; var Allow: Boolean);
begin
  if TheClient.FUserName = 'TestUser' and Token <> MetaData then Allow
  := False;
  // If you want to restrict to server methods only, the token to look
```

```
    for is MetaData
  end;
```

See Also
OnBeforeServerMethod

1.8.2.4.3.7 TAstaServerSocket.OnBeforeServerMethod

TAstaServerSocket

Declaration

```
property OnBeforeServerMethod: TAstaBeforeServerMethodEvent;
type TAstaBeforeServerMethodEvent = procedure(Sender: TObject; TheClient:
  UserRecord; MethodName: string; var Allow: Boolean) of object;
```

Description

This event is fired before any server method is run. It allows you a central place to make security restrictions on which server methods can be run by which users.If you set allow to False, an exception will be raised on the client. As an example, take a user "TestUser" and you want to restrict it to only run a particular server method "GetUserList":

```
procedure TfrmMain.AstaServerBeforeServerMethod(Sender:
  TObject;TheClient: UserRecord;
MethodName: string; var Allow : Boolean)
begin
  if TheClient.FUserName = 'TestUser' and MethodName <> 'GetUserList'
  then
    Allow := False;
end;
```

See Also
OnBeforeProcessToken

1.8.2.4.3.8 TAstaServerSocket.OnChatLine

TAstaServerSocket

Declaration

```
property OnChatLine: TAstaServerSendStringsEvent;
```

Description

The OnChatLine event allows you to take action when chat messages are passed through the sever. The following code, for instance, displays the messages in a memo at the server.

```
procedure TForm1.AstaServerSocket1ChatLine(Sender: TObject; S: string);
begin
  ReceiveMemo.Lines.Add(S);
end;
```

You could stream the memo to a file or perhaps save it to a database on a daily basis.

1.8.2.4.3.9 TAstaServerSocket.OnCheckOutSession

TAstaServerSocket

Declaration

```
property OnCheckOutSession: TServerCheckOutSession;
TServerCheckOutSession = procedure (Sender: TObject; At: TAstaThread;
  ClientSocket: TCustomWinSocket; Session: TComponent) of object;
```

Description

This event will fire whenever a Database Session is "checked Out" this event will fire. If there is any housekeeping you want to do on a DataModule you can code this event. The Session:TComponent parameter will be the main Datamodule created in ThreadedDBSupplySession when the server starts up in Pooled Threading.

```
procedure TForm1.AstaServerSocket1CheckOutSession(Sender: TObject;
  At: TAstaThread; ClientSocket: TCustomWinSocket; Session: TComponent);
begin
  with TAstaDataModule(Session) do begin
    //do what you want to do here on the datamodule
  end;
end;
```

1.8.2.4.3.10 TAstaServerSocket.OnClientAuthenticate

TAstaServerSocket

Declaration

```
property OnClientAuthenticate: TAstaClientAuthenticateEvent;
type TAstaClientAuthenticateEvent = procedure (Sender: TObject; Client:
  TUserRecord; Name, Password, AppName: string; ClientParamList:
  TAstaParamList; var LoginVerified: Boolean; var PersistentSession:
  TComponent) of object;
```

Description

The OnClientAuthenticate event is used to:

- Authenticate Remote Users
- Send back optional information using the ClientParamList
- Optionally allow for a persistent Session to be assigned to a particular user when using tmSmartThreading

Every remote client logging in will send a Username and Password to the Asta server as well as extra items that were added to the TAstaClientSocket.ClientSocketParams which are merged into the UserRecord.ParamList. Set the LoginVerified parameter to authenticate the client and optionally fill the ClientParamList with any values you want passed back that will appear in the TAstaClientSocket.OnLoginParamsAttempt event. If you want the user to have a specific Database Session, create a DataModule and assign it to the PersistentSession:TComponent event. Users can move between Pooled and Persisent Sessions using ASTA Smart Threading and this event is also used for that purpose.

Example

```
procedure TServerDM.ServerSocketClientAuthenticate(Sender: TObject;
  Client: TUserRecord; UName, Password, AppName: string;
  ClientParamList: TAstaParamList; var LoginVerified: Boolean; var
  PersistentSession: TComponent);
begin
  //Create a datamodule for PersisentSession if you are running
  tmSmartThreading and you want any specific user to have their own
  session

  LoginVerified := True;
end;
```

1.8.2.4.3.11 TAstaServerSocket.OnClientConnect

TAstaServerSocket

Declaration
```
property OnClientConnect: TSocketNotifyEvent;
```

Description
Occurs when a client socket completes a connection accepted by the server socket.. For more information see the TServerSocket.OnClientConnect in the Delphi help file.
The ClientSocket has not been added to the UserList when this event occurs so if you want to send a message to a client socket from this event you must call EarlyRegisterSocket.

1.8.2.4.3.12 TAstaServerSocket.OnClientDBLogin

TAstaServerSocket

Declaration
```
property OnClientDBLogin: TAstaLoginEvent;
type TAstaDBLoginEvent = procedure(Sender: TObject; TheUserName,
  ThePassword, TheAppName: string; var Verified: Boolean; var ADataBase:
  TComponent) of object;
```

Description
Supported in ASTA 3 for ASTA 2 compatibility but the new OnClientAuthenticate event is recommended.

OnClientDBLogin is implemented when you want to have your remote users use their database user name and password or to have them to have access to a different server database, and is used primarily with the PersistentSession threading model. For normal logins use the OnClientLogin. If you want to validate IP addresses before the login use OnClientValidate.

Note: The Verified parameter returns the results of the attempt to the client. If you verification attempt succeeds then set Verified to True, if it fails, set Verified to False. If you want the client to be automatically disconnected then set DisconnectClientOnFailedLogin to True in which case nothing will be returned to the client but they will be immediately disconnected from the server.

1.8.2.4.3.13 TAstaServerSocket.OnClientExceptionMessage

TAstaServerSocket

Declaration

```
property OnClientExceptionMessage: TServerExceptionEvent;
type TServerExceptionEvent = procedure (Sender: TObject; ClientSocket:
  TCustomWinSocket; var ErrorMsg: string) of object;
```

Description

ASTA servers are coded so that typical database requests from clients that raise exceptions are sent back using SendExceptionToClient. By coding the OnClientExceptionMessage you can intercept any exceptions raised by your database and re-word them for your client application.

1.8.2.4.3.14 TAstaServerSocket.OnClientLogin

TAstaServerSocket

Declaration

```
property OnClientLogin: TAstaLoginEvent;
type TAstaLoginEvent = procedure(Sender: TObject; UserName, Password,
  AppName: string; var Verified: Boolean) of object;
```

Description

Assign code to this event handler if you wish to provide login routines using the AstaClientSocket's AutoLoginDlg property. If you wanted to compare an AstaClient's user name and password input against a list of valid user names and passwords that you maintain, this is the place to do it. The TAstaClientSocket has a corresponding OnLoginAttempt event.

Note: The Verified parameter returns the results of the attempt to the client. If you verification attempt succeeds then set Verified to True, if it fails, set Verified to False.

The following code simply adds the user name, password and the application name to a memo on the server.

```
procedure TForm1.AstaServerSocket1ClientLogin(Sender: TObject; UserName,
  Password, AppName: string; var Verified: Boolean);
begin
  mLogin.Lines.Add('UserName: ' + UserName + ' Password: ' +
    Password + ' Application Name: ' + AppName);
  //Return a successful login attempt
  Verified := True;
end;
```

Note: This event is supported in ASTA 3 for ASTA 2 compatibility but OnClientAuthenticate is recommended.

1.8.2.4.3.15 TAstaServerSocket.OnClientValidate

TAstaServerSocket

Declaration

```
property OnClientValidate: TValidateSocketEvent;
type TValidateSocketEvent = procedure(Sender: TObject; ClientSocket:
```

```
  TCustomWinSocket; var IsValid: Boolean) of object;
```

Description
Use this event if you want to validate remote clients on connection. This occurs before the OnClientAuthenticate at connect time and allows a reverse lookup of IP addresses to be used for increased security.

Example

```
procedure TForm1.ServerClientValidate(Sender: Object; ClientSocket:
  TCustomWinSocket; var IsValid: Boolean);
begin
  IsValid := ClientSocket.RemoteAddress = '127.0.0.1';
end;
```

Security Issues

1.8.2.4.3.16 TAstaServerSocket.OnCodedMessage

TAstaServerSocket

Declaration

```
property OnCodedMessage: TAstaServerSendcodedMessageEvent;
type TAstaServerSendCodedMessage = procedure (Sender: TObject;
  Clientsocket: TCustomWinSocket; Msgid, MsgToken: Integer) of object;
```

Description
The OnCodedMessage event handler allows you to take customized application action according your own custom messages that have been sent via an AstaClientSocket's SendCodedMessage method.

The following code shows how the server might respond upon receiving certain messages.

```
procedure TForm1.AstaServerSocket1CodedMessage(Sender: TObject;
  ClientSocket: TCustomWinSocket; MsgID: Integer; Msg: string) of
  object;
var
  MS: TMemoryStream;
begin
  case MsgID of
    1700: AstaServerSocket1.SendCodedMessage(ClientSocket, MsgID,
            UpperCase(Msg));
    1750: AstaServerSocket1.SendCodedMessage(ClientSocket, MsgID,
            LowerCase(Msg));
    2000: begin
            MS := TMemoryStream.Create;
            mSelectSend.Lines.SaveToStream(MS);
            AstaServerSocket1.SendCodedStream(ClientSocket, 2000, MS);
            MS.Free;
          end;
  end;
end;
```

1.8.2.4.3.17 TAstaServerSocket.OnCodedParamList

TAstaServerSocket

Declaration
```
property OnCodedParamList: TAstaServerParamEvent;
```

Description
The OnCodedParam event handler allows you to take customized application action according your own custom messages that have been sent via an AstaClientSocket's SendCodedParamList method.
Note: when the AstaclientSocket uses SendGetCodedParamList the incoming parameters will be automatically returned to the client by ASTA, so it is up to you to clear the parameters and add any parameters you would like to be returned to the client.

There is a new OnOutCodedParamList that now allows for OnCodedParamList to be used for Async calls from a client and OnOutCodedParamlist to be used for blocking calls.

The following code shows how the server might respond upon receiving certain messages.

```
procedure TForm1.AstaServerSocket1CodedParamList(Sender: TObject;
  ClientSocket: TCustomWinSocket; MsgID: Integer;
  Params: TAstaParamList);
var
  I: Integer;
  M: TMemorystream;
  dm: TAstaDataModule;
begin
  LogIt('Coded Params for Msgid:' + IntToStr(Msgid));
  for I := 0 to Params.Count - 1 do

    case Params[I].DataType of
      ftBytes, ftBlob, ftGraphic:
        begin
          LogIt(' Params[' + IntToStr(I) + '] -> Blob Size ' +
            IntToStr(Length(Params[I].AsString)))
        end;
    else
      LogIt(' Params[' + IntToStr(I) + '] ->' + Params[I].AsString);
    end;

    case Msgid of
       100: if (Pos('zip', Params[I].Name) > 0) and
              (Params[I].DataType = ftBlob) then
            begin
              M := TMemoryStream(Params[I].AsStream);
              M.SaveToFile(Params[I].Name);
              LogIt('Save to disk: ' + Params[I].Name);
              M.Free;
            end;
      1500: begin
              Params.Clear;
              AstaServerSocket1.ThreadedDBUtilityEvent(ClientSocket,
                ttSelect, TestUtilEvent, Params);
            end;
```

```
      1550: AstaServerSocket1.ThreadedDBUtilityEvent(ClientSocket,
              ttSelect, TestUtilSQLEvent, Params);
        1600: begin
                Params.Clear;
                AstaServerSocket1.ThreadedUtilityEvent(ClientSocket,
                  TestPlainUtilEvent, Params);
              end;
        1800: begin
                ChangeThreadingModel(ClientSocket,
                  TAstaThreadModel(Params[0].AsInteger),
                  Params[1].AsInteger);
              end;
        1801: begin
                Params.Clear;
                Params.FastAdd(ord(AstaServerSocket1.ThreadingModel));
                Params.FastAdd(AstaServerSocket1.DataBaseSessions);
              end;
        1900: begin
                //this requires persisentsessions threading
                dm := TAstaDataModule(AstaServerSocket1.SessionList.
                  GetSessionFromSocket(ClientSocket, ''));
                if dm <> nil then LogIt(dm.Name);
              end;
        2001: begin
                if FileExists(Params[0].AsString) then begin
                  M := TMemoryStream.Create;
                  M.LoadFromFile(Params[0].AsString);
                  M.Position := 0;
                  LogIt(Params[0].AsString + ' ' + IntToStr(M.Size));
                  Params.Clear;
                  params.Fastadd('');
                  Params[0].AsStream := M;
                  M.Free;
                end
                else
                  Params.Clear;
              end;
      end;
    end;
  end;
```

1.8.2.4.3.18 TAstaServerSocket.OnCodedStream

TAstaServerSocket

Declaration

```
property OnCodedStream: TAstaServerStreamEvent;
```

Description

The OnCodedStream event is similar to the OnCodedMessage event, the MsgID allows you to create your own protocol codes and then take custom action corresponding to that code. The OnCodedStream event handler is for handling streams and blobs. See the SendCodedStream method.

The following code takes a stream from the server, and as dictated by the user-defined 850 protocol code, it displays the stream in a memo field, then writes it to a file.

```
procedure TForm1.AstaServerSocket1CodedStream(Sender: TObject;
```

```
    ClientSocket: TCustomWinSocket; MsgID: Integer; MS: TMemoryStream);
  begin
    case MsgID of
      850: begin
             mServerMemo.Lines.Clear;
             mServerMemo.Lines.LoadFromStream(MS);
             mServerMemo.Lines.SaveToFile('ServerFile.txt');
           end;
    end;
  end;
```

1.8.2.4.3.19 TAstaServerSocket.OnCompress

TAstaServerSocket

Declaration

```
property OnCompress: TAstaServerStringEvent;
```

Description

This event handler allows you to add code to compress messages before they are sent across the network. It should be noted that compression/decompression routines will have performance considerations. Compressing large messages before they are sent over slow network connections is sensible. Compressing small messages before transmission over fast LANs may actually impede performance. It might take longer to compress/decompress the data then it takes for the data to travel across the network. NOTE: If you add compression routines, you must add decompression routines. You must place your code in the appropriate event handlers (OnCompress and OnDecompress) of the AstaClientSocket and the AstaServerSocket.

ASTA Compression

1.8.2.4.3.20 TAstaServerSocket.OnCustomAutoIncFormat

TAstaServerSocket

Declaration

```
property OnCustomAutoIncFormat: TAstaServerCustomAutoIncFormat;
type TAstaServerCustomAutoIncFormat = procedure(Sender: TObject;
  TheDatabase, TheTableName: string; var TheAutoIncrementValue: string);
```

Description

Allows for the customization of the AutoIncValue.

1.8.2.4.3.21 TAstaServerSocket.OnDecompress

TAstaServerSocket

Declaration

```
property OnDecompress: TAstaServerStringEvent;
```

Description

This event handler allows you to add code to decompress messages after they have been sent, in an encrypted form, across the network. It should be noted that compression/decompression routines will have performance considerations. Compressing large messages before they are sent over slow network connections is sensible. Compressing small messages before transmission over fast LANs may actually impede

performance. It might take longer to compress/decompress the data then it takes for the data to travel across the network.

NOTE: If you add compression routines, you must add decompression routines. You must place your code in the appropriate event handlers (OnCompress and OnDecompress) of the AstaClientSocket and the AstaServerSocket.

ASTA Compression

1.8.2.4.3.22 TAstaServerSocket.OnDecrypt

TAstaServerSocket

Declaration

```
property OnDecrypt: TAstaServerStringEvent;
type TAstaServerStringEvent = procedure(Sender: TObject; TheClient:
  TUserRecord; var S: string) of object;
```

Description

The OnDecrypt event handler allows you to add code for decrypting an encrypted message.

Note: If you add encryption routines to the OnEncrypt event handlers, then you must add decryption routines to the OnDecrypt event handlers. You must place your code in the appropriate event handlers (OnEncrypt and OnDecrypt) of the AstaClientSocket and the AstaServerSocket.

Example

```
procedure TForm1.AstaServerSocket1Decrypt(Sender: TObject; var S:
string);
begin
  S := SimpleDecrypt(S);
end;
```

1.8.2.4.3.23 TAstaServerSocket.OnEncrypt

TAstaServerSocket

Declaration

```
property OnEncrypt: TAstaServerStringEvent;
type TAstaServerStringEvent = procedure(Sender: TObject; TheClient:
  TUserRecord; var S: string) of object;
```

Description

The OnEncrypt event handler allows you to add code for encrypting a message before transmitting it across a network.
Note: If you add encryption routines to the OnEncrypt event handlers, then you must add decryption routines to the OnDecrypt event handlers and of course the Server and Client must use the same Encryption scheme. You must place your code in the appropriate event handlers (OnEncrypt and OnDecrypt) of the AstaClientSocket and the AstaServerSocket. The Sender:TObject on the AstaServer is a

ClientSocket:TCustomWinSocket. Using the ClientSocket you are able to access the UserRecord to provide stronger encyrption with multiple keys.

```
procedure TForm1.AstaServerSocket1Encrypt(Sender: TObject; var S:
  string);
begin
  // the Sender is the  ClientSocket:TCustomWinSocket
  S := MyEncryptRoutine(S);
end;
```

There are many third party components that can be used to secure your ASTA applications. Because of US export laws ASTA cannot supply strong encryption routines as part of ASTA.

1.8.2.4.3.24 TAstaServerSocket.OnExecRowsAffected

TAstaServerSocket

Declaration

```
property OnExecRowsAffected: TAstaServerRowsAffectedEvent;
type TAstaServerRowsAffectedEvent = procedure(Sender: TObject;
  ClientSocket: TCustomWinSocket; AQuery: TComponent; var RowsAffected:
  Integer) of object;
```

Description

Previously ASTA users had to code the OnExecSQLParamList of the AstaServerSocket to send back a custom message to the AstaClientSocket to retrieve this information.

See also TAstaClientDataSet.RowsAffected.

Example

```
procedure TForm1.AstaServerSocket1ExecRowsAffected(Sender: TObject;
ClientSocket: TCustomWinSocket; AQuery: TComponent; var RowsAffected:
Integer);
begin
  try
    RowsAffected := TQuery(AQuery).RowsAffected;
  except
    LogException(ClientSocket, Exception(ExceptObject).Message, True);
  end;
end;
```

1.8.2.4.3.25 TAstaServerSocket.OnExecSQLParamList

TAstaServerSocket

Declaration

```
type TAstaServerExecSQLParamListEvent = procedure(Sender: TObject;
  ClientSocket: TCustomWinSocket; AQuery: TComponent;
  DataBaseStr, SQLString: string; var TheResult: Boolean;
  var Msg: string; ParamList: TAstaParamList) of object;
```

Description

This is the event coded that allows Exec statements to work for ASTA clients. ASTA clients will send in parameterized queries that appear as the SQLString and ParamList parameters. The server must be able to handle parameterized queries in order to be able to update and insert blobs and memos. To support the RowsAffected property of the AstaClientDataSet the OnExecRowsAffected Event of the AstaServer must be coded.

See Coding ASTA Servers for steps to create an ASTA server.

Here is an example from the AstaBDEServer. Depending on the threading model in effect, the AQuery parameter will be supplied by the ThreadedDBSupplyQueryEvent. Since ASTA suports any type of component for use with ASTA servers, the example below type casts the AQuery parameter as a BDE TQuery since this example was from the AstaBDEServer.

```
procedure TForm1.AstaServerSocket1ExecSQLParamList(Sender: TObject;
  ClientSocket: TCustomWinSocket; AQuery: TComponent; DataBaseStr,
  SQLString: string; var TheResult: Boolean; var Msg: string;
  ParamList: TAstaParamList);
var
  I: Integer;
begin
  try
    TheResult := False; //we need to know if the exec is successful
    TQuery(AQuery).SQL.Clear;
    TQuery(AQuery).SQL.Add(SqlString);
    for I := 0 to ParamList.Count - 1 do begin
      if ParamList[i].IsNull then begin
        TQuery(AQuery).Params[i].DataType := ParamList[i].DataType;
        TQuery(AQuery).Params[i].Clear;
        TQuery(AQuery).Params[i].Bound := True;
      end
      else
        case ParamList[i].DataType of
          ftString: TQuery(AQuery).Params[i].AsString :=
                      ParamList[i].AsString;
          ftInteger,
          ftSmallint,
          ftWord: TQuery(AQuery).Params[i].AsInteger :=
                    ParamList[i].AsInteger;
          ftFloat: TQuery(AQuery).Params[i].AsFloat :=
                     ParamList[i].AsFloat;
          ftBoolean: TQuery(AQuery).Params[i].AsBoolean :=
                       ParamList[i].AsBoolean;
          ftTime: TQuery(AQuery).Params[i].AsTime :=
                    ParamList[i].AsTime;
          ftDate,
          ftDatetime: TQuery(AQuery).Params[i].AsDateTime :=
                        ParamList[i].AsDateTime;
```

```
    ftBlob,
            ftGraphic: TQuery(AQuery).Params[i].SetBlobData(
                         pchar(ParamList[i].AsString),
                         Length(ParamList[i].AsString));
            ftMemo: TQuery(AQuery).Params[i].AsMemo :=
                      ParamList[i].AsString;
          end;
      end;
      TQuery(AQuery).ExecSQL;
      TheResult := True;
    except
      begin
        Msg := EDataBaseError(ExceptObject).Message;
        LogIt('Exec SQL Error->' + Msg);
      end;
    end;
  end;
```

1.8.2.4.3.26 TAstaServerSocket.OnFetchBlobEvent

TAstaServerSocket

Declaration

```
property OnFetchBlobEvent: TAstaFetchBlobEvent;
type TAstaFetchBlobEvent = procedure(Sender: TObject;
  ClientSocket: TCustomWinsocket; AQuery: TComponent;
  DataBaseStr, TableName, FieldName, WhereString: string;
  var M: TMemoryStream) of object;
```

Description

In order to to fetch a single blob from an ASTA server this event must be coded. The following is an example from the AstaBDEServer.

See Coding ASTA Servers for steps to create an ASTA server.

```
procedure TForm1.AstaServerSocket1FetchBlobEvent(Sender: TObject;
  ClientSocket: TCustomWinSocket; AQuery: TComponent; DataBaseStr,
  TableName, FieldName, WhereString: string; var M: TMemoryStream);
var
  S: string;
begin
  try
    if Query.Active then Query.Close;
    TQuery(AQuery).SQL.Clear;
    if MainDataBase.IsSQLBased then S := 'SELECT ' + FieldName +
      ' FROM ' + Tablename + ' ' + WhereString
    else
      //Paradox needs the " for spaces in table names;
      S := 'SELECT ' + TableName + '."' + FieldName + '" FROM ' +
        TableName + ' ' + WhereString;
    TQuery(AQuery).SQL.Add(S);
    TQuery(AQuery).Open;
    if TQuery(AQuery).Recordcount > 0 then begin
      case TQuery(AQuery).FieldByName(FieldName).DataType of
        ftMemo: TMemoField(TQuery(AQuery).FieldByName(
                  FieldName)).SaveToStream(M);
        ftBlob,
        ftGraphic: TBlobField(TQuery(AQuery).FieldbyName(
                     FieldName)).SaveToStream(M);
      end;
    end;
  except
    LogException(ClientSocket, Exception(exceptObject).Message, True);
  end;
end;
```

1.8.2.4.3.27 TAstaServerSocket.OnFetchMetaData

TAstaServerSocket

Declaration

```
type TAstaServerMetaDataEvent = procedure(Sender: TObject;
  ClientSocket: TCustomWinSocket; MetaRequest: TAstaMetaData;
  DataBaseStr, Arg1, Arg2: string) of object;
```

Description

ASTA clients require information about the server database, and coding the OnFetchMetaData event is required for ASTA clients to be able to fetch tablenames, fieldnames, index information, foreign key information, stored procedure names and columns and other assorted server database information. AstaClientDataSets trigger this server side event using their MetaDataRequest property.

The OnFetchMetaData event is currently NOT threaded and will return the AstaServerSocket.MetaDataSet to ASTA clients.

The following is an abridged version of the OnFetchMetaData event as coded on the AstaBDEServer. An important point is that whatever request that comes in from the client, the MetaData property is set on the TAstaServerSocket so that the correct information is streamed back to ASTA clients. The AstaBDEServer uses the ASTA component AstaBDEInfoTable to retrieve BDE information and the AstaODBCInfoDataSet to retrieve meta data information using components from www.odbcexpress.com. Source is available for these two ASTA units on request from ASTA registered users.

```
procedure TForm1.AstaServerSocket1FetchMetaData(Sender: TObject;
  ClientSocket: TCustomWinSocket;  MetaRequest: TAstaMetaData;
  DataBaseStr, Arg1, Arg2: string);
var
  i: Integer;
begin
  try
    case MetaRequest of
      mdIndexes, mdPrimeKeys, mdForeignKeys:
        begin
          LogIt('Index Info for ' + Arg1);
          IndexInfoTable.Close;
          IndexInfoTable.TableName := Trim(Arg1);
          AdjustBDEInfoDatabaseName(DataBaseStr, IndexInfoTable);
          IndexInfoTable.Active := True;
          if AstaDataSet1.Active then AstaDataSet1.Empty;
          AstaDataSet1.CloneFieldsFromDataSet(IndexInfoTable,
            False, False);
          IndexDataTransfer(IndexInfoTable, AstaDataSet1);
          AstaServerSocket1.MetaDataSet := AstaDataSet1;
        end;
      mdStoredProcColumns:
        begin
          //ODBC and the BDE create different column names and types.
          //When returning param info the BDEInfoToStoredProcAstaInfoTable
          //call uses a standard datadet (AstaStoredProcColumnDataSet)
          //that the ASTA client.SQLWorkbench can use.
          //Other backends need to populate the fields in
          //AstaStoredProcColumnDataSet
          TableDataSet.Close;
          TableDataSet.BDEInfo := BDEStoredProcParams;
          AdjustBDEInfoDatabaseName(DataBaseStr, TableDataSet);
          TableDataSet.TableName := Arg1;
          TableDataSet.Open;
          BDEInfoToStoredProcAstaInfoTable(TableDataSet,
            AstaStoredProcColumnDataSet);
          AstaServerSocket1.MetaDataSet :=
            AstaStoredProcColumnDataSet;
          LogIt('Stored Proc Param Info for ' + Arg1);
        end;
      mdStoredProcs:
```

```
begin
          {$ifdef StoredProcProblemo}
          PopulateStoredProcNames;
          {$else}
          AstaServerSocket1.MetaDataSet := TableDataSet;
          TableDataSet.Close;
          TableDataSet.BDEInfo := BDEStoredProcedures;
          AdjustBDEInfoDatabaseName(DataBaseStr, TableDataSet);
          TableDataSet.TableName := '';
          TableDataSet.Open;
          {$endif}
          LogIt('Stored Procedure List');
        end;
      mdDBMSName:
        begin
          LogIt('DBMS Info Request');
          AstaServerSocket1.MetaDataSet := DBMSDataSet;
        end;
      mdServerDataSets:
        begin
          LogIt('Server DataSet Info Request');
          AstaServerSocket1.MetaDataSet := ServerDataSets;
        end;
    else begin
      LogIt('Table Info');
      TableDataSet.Close;
      AdjustBDEInfoDatabaseName(DataBaseStr, TableDataSet);
      TableDataSet.TableName := '';
      TableDataSet.BDEInfo := BDEOpenTables;
      TableDataSet.Open;
      AstaServerSocket1.MetaDataSet := TableDataSet;
    end;
  except
    LogException(ClientSocket, 'Stored MetaData Error: ' +
      EDataBaseError(ExceptObject).Message, True);
  end;
end;
```

1.8.2.4.3.28 TAstaServerSocket.OnInsertAutoIncrementFetch

TAstaServerSocket

Declaration

```
property OnInsertAutoIncrementFetch: TAstaAutoIncrementInsertEvent;
type TAstaAutoIncrementInsertEvent = procedure(Sender: TObject;
  ClientSocket: TCustomWinSocket; AQuery: TComponent; TheDatabase,
  TheTable, TheAutoIncField: string; var AutoIncrementValue: Integer) of
  object;
```

Description

The OnInsertAutoIncrementFetch event must be coded if you wish to retrieve auto increment field values at the client. This event is used in conjunction with the TAstaClientDataSet's AutoIncrementField and RefetchOnInsert properties.

```
procedure TForm1.AstaServerSocket1InsertAutoIncrementFetch(Sender:
  TObject; ClientSocket: TCustomWinSocket; AQuery: TComponent;
  TheDatabase, TheTable, TheAutoIncField string; var AutoIncrementValue:
  Integer);
begin
  //put your code in here.
  //Something like
  with aQuery as TQuery do begin
    SQL.Add('select max(' + TheAutoIncField + ') from ' + TheTable);
    Open;
    TheAutoIncrementValue := FieldByName(TheAutoIncField).AsInteger;
```

```
    end;
  end;
```

The above code is specific to your database. For Access, Paradox and Interbase 5.X the above code should work in calling MAX to get the last value used by the server for an auto increment field. For MS SQL Server, Sybase, and SQL Anywhere you would need to write code to access the @@identify variable. The AstaODBC server is already coded to identify these databases and make the appropriate call.

1.8.2.4.3.29 TAstaServerSocket.OnInstantMessage

Declaration

Description

1.8.2.4.3.30 TAstaServerSocket.OnOutCodedDBParamList

TAstaServerSocket

Declaration

```
property OnOutCodedDBParamList: TAstaServerOutDBParamEvent;
type TAstaServerOutDBParamEvent = procedure(Sender: TObject; ClientSocket:
  TCustomWinSocket; DBDataModule: TComponent; MsgID: integer; InParams,
  OutParams: TAstaParamList) of object;
```

Description

This new event is available which will be called in a thread with a datamodule handed off to you with any and all of your database components. You can also create and destroy DB components on the fly.

Just Grab the incoming Params and fill up the OutParams and they will be transported back to the client in a thread. if an exception occurs it will be raised at the client. or you can raise it yourself on the server to be transported back to the client.

```
  procedure TServerDM.ServerSocketOutCodedDBParamList(Sender: TObject;
    ClientSocket: TCustomWinSocket; DBDataModule: TComponent; MsgID:
    Integer; InParams, OutParams: TAstaParamList);
  begin
    with dbDataModule as TAstaDataModule do begin
     Query.SQL.Text := 'select * from customers';
     Query.Open;
     OutParams.FastAdd('Count', Query.Recordcount);
    end;
  end;
```

See also: TAstaClientSocket.SendGetCodedDBParamList

1.8.2.4.3.31 TAstaServerSocket.OnOutCodedParamList

TAstaServerSocket

Declaration

```
property OnOutCodedParamList: TAstaServerOutParamEvent;
type TAstaServerOutParamEvent = procedure(Sender: TObject; ClientSocket:
  TCustomWinSocket; MsgID: integer; InParams, OutParams: TAstaParamList) of
```

```
  object;
```

Description
OnOutCodedparamlist supports SendGetCodedParamList calls from the AstaClientSocket. These calls "block or wait". On the server, the Incoming Params from the client are passed in as inParams. On the server, just use any of the InParams and fill up the OutParams with any value you want and they will be returned to the client in a thread.

Example

```
procedure TServerDM.ServerSocketOutCodedParamList(Sender: TObject;
ClientSocket: TCustomWinSocket; MsgID: Integer; InParams, OutParams:
TAstaParamList);
begin
  OutParams.FastAdd('ServerTime', Now);
end;
```

1.8.2.4.3.32 TAstaServerSocket.OnProviderCompleteTransaction

TAstaServerSocket

Declaration

```
property OnProviderCompleteTransaction:
  TAstaProviderCompleteTransactionEvent;
type TAstaProviderCompleteTransactionEvent = procedure(Sender: TObject;
  TransactionCommitted: Boolean; const Providers: array of TAstaProvider)
  of object;
```

Description
When using AstaSocket.SendProviderTransaction from remote clients to send multiple ClientDataSets, connected to a TAstaProvider, to the server in one server round trip, the OnProviderCompleteTransaction event will fire after all operations from all providers are successfull or at first failure. The TransactionCommited:Boolean param signifies the success of the transaction.

This is useful if you want to hand craft your own Provider broadcasts with ASTA Provider manual broadcast options.

See also: Provider Bite at the Apple

1.8.2.4.3.33 TAstaServerSocket.OnProviderConvertParams

TAstaServerSocket

Declaration

```
type TAstaProviderConvertParams = procedure(Sender :TObject;
  D: TDataSet; P: TAstaParamList; var Converted: Boolean) of object;
```

Description
This method along with the OnProviderSetParams method of the TAstaServerSocket allows TAstaProviders using TDataSets that do not have a TParams,
Params property to be supported by ASTA.

Example

```
procedure TForm1.AstaServerSocket1ProviderConvertParams(Sender: TObject;
  D: TDataSet; P: TAstaParamList; var Converted: Boolean);
var
  i: Integer;
begin
  with D as TDBisamQuery do begin
    for i := 0 to Params.Count - 1 do
      P.CreateParam(Params[i].DataType, Params[i].Name,
        AstaParamList.ptInput);
  end;
end;
```

1.8.2.4.3.34 TAstaServerSocket.OnProviderSetParams

TAstaServerSocket

Declaration

```
type TAstaProviderSetParams = procedure(Sender: TObject;
  D: TDataSet; P: TAstaParamList) of object;
```

Description

This method along with the OnProviderConvertParams method of the TAstaServerSocket allows TAstaProviders using TDataSets that do not have a TParams, Params property to be supported by ASTA.

Example

```
procedure TForm1.AstaServerSocket1ProviderSetParams(Sender: TObject; D:
  TDataSet; P: TAstaParamList);
var
  i: Integer;
begin
  with D as TDbisamQuery do begin
    Close;
    Prepare;
    for i := 0 to P.Count - 1 do
      Params[i].Value := P[i].Value;
  end;
end;
```

1.8.2.4.3.35 TAstaServerSocket.OnServerQueryGetParams

TAstaServerSocket

Declaration

Description

1.8.2.4.3.36 TAstaServerSocket.OnServerQuerySetParams

TAstaServerSocket

Declaration

```
property OnServerQuerySetParams: TAstaServerQuerySetParams;
type TAstaServerQuerySetParams = procedure(Sender: TObject;
  ClientSocket:TCustomWinSocket; Query:TComponent; ParamList:
```

```
  TAstaParamList) of object;
```

Description
When parameterized queries are sent from remote ASTA clients the originate from the TAstaClientDataSet.Params property which is type TAstaParamList. Since ASTA supports any Delphi 3rd party DataSet Components on ASTA servers, those AstaParamLists need to be translated to the param type supported by the Delphi 3rd party Database component used on the server. On Both Select and Exec SQL, a typical ASTA server implementation will call ServerSetSQLParams which internally uses the OnServerQuerySetParams event.

See also:
OnSubmitSQLParams
OnProviderSetParams

Below is an example of this event as coded on the AstaADOServer:

```
procedure TAstaADOPlugin.ADOServerQuerySetParams(Sender: TObject;
  ClientSocket: TCustomWinSocket; Query: TComponent; ParamList:
  TAstaParamList);
var
  i: Integer;
  m: TMemoryStream;
  AdoParam: TParameter;
begin
  for I := 0 to ParamList.Count - 1 do begin
    AdoParam :=
  TAdoQuery(Query).Parameters.FindParam(ParamList[i].Name);
    if ParamList[i].IsNull and (ParamList[i].ParamType <>
  AstaParamList.ptOutput) then begin
      ADOParam.DataType := ParamList[i].DataType;
      ADOParam.Value := VarNull;
     end else
     if paramlist[i].ParamType <> AstaParamList.ptOutput then
       case ParamList[i].DataType of
         ftString: ADOParam.Value := ParamList[i].AsString;
         ftInteger, ftSmallint, ftWord, ftAutoInc: ADOParam.Value :=
  ParamList[i].AsInteger;
         ftFloat: ADOParam.Value := ParamList[i].AsFloat;
         ftBoolean: ADOParam.Value := ParamList[i].AsBoolean;
         ftTime: ADOParam.Value := ParamList[i].AsTime;
         ftDate, ftDateTime: ADOParam.Value := ParamList[i].AsDateTime;
         ftGraphic:
           begin
             m := TMemoryStream(ParamList[i].AsStream);
             try
               ADOParam.LoadFromStream(m, ftBlob);
             finally
               m.free;
             end;
           end;
         ftBlob:
           begin
             m := TMemoryStream(ParamList[i].AsStream);
             try
               ADOParam.LoadFromStream(m, ftBlob);
             finally
               m.Free;
```

```
      end;
              end;
            ftMemo:
              begin
                m := TMemoryStream(ParamList[i].AsStream);
                try
                  ADOParam.LoadFromStream(m, ftMemo);
                finally
                  m.Free;
                end;
              end;
          end;
        end;
    end;
```

1.8.2.4.3.37 TAstaServerSocket.OnShowServerMessage

TAstaServerSocket

Declaration

```
type TAstaServerShowMessageEvent = procedure(Sender: TObject;
  Msg: string) of object;
```

Description

This event is used internally by ASTA servers to record server activity. You may want to have this write to a log file. Note there is a performance penalty of course for writing to a TMemo on an ASTA server so you should be able to turn this off in production. To control logging on an ASTA server code this event and use the LogOn property.

1.8.2.4.3.38 TAstaServerSocket.OnStatelessClientCheckIn

Declaration

Description

1.8.2.4.3.39 TAstaServerSocket.OnStatelessClientCookie

Declaration

Description

1.8.2.4.3.40 TAstaServerSocket.OnStateLessPacketEvent

Declaration

Description

1.8.2.4.3.41 TAstaServerSocket.OnStoredProcedure

TAstaServerSocket

Declaration

```
property OnStoredProcedure: TAstaServerStoredProcEvent;
type TAstaServerStoredProcEvent = procedure(Sender: TObject; ClientSocket:
  TCustomWinSocket; AQuery: TComponent; DataBaseStr, StoredProcName:
  string; ClientParams: TAstaParamList; NoResultSet: Boolean) of object;
```

Description

This event must be coded to support StoredProcedures using ASTA. The client will send in the StoredProcName and a TAstaParamList that will contain the parameters for the stored procedure. If parameters are to be returned, the the ClientParams must be updated. Note the NoResultSet:Boolean which is required for StoredProcedures that do not return a result set. The AstaClientDataSet must call ExecSQL when invoking StoredProcedures to that do not return a result set.

See Coding ASTA Servers for steps to create an ASTAServer.

The following is an example from the AstaBDEServer showing how this method can be coded.

```
procedure TForm1.AstaServerSocket1StoredProcedure(Sender: TObject;
  ClientSocket: TCustomWinSocket; AQuery: TComponent; DataBaseStr,
  StoredProcName: string; ClientParams: TAstaParamList; NoResultSet:
  Boolean);
var
  I: Integer;
  P: TParam;
begin
  try
    if TStoredProc(Aquery).Active then
      TStoredProc(AQuery).Active := False;
    for i := 0 to ClientParams.Count - 1 do
      TStoredProc(AQuery).ParambyName(ClientParams[i].Name).Value
  := ClientParams[i].Value;
    //TStoredProc(AQuery).Prepare; prepare is now called in the
    //setup stored proc call
    if NoResultSet then
      TStoredProc(Aquery).ExecProc
    else
      TStoredProc(AQuery).Active := True;
    ClientParams.Clear;
    for i := 0 to TStoredProc(AQuery).Params.Count - 1 do begin
      case TStoredProc(AQuery).Params[i].ParamType of
        db.ptOutPut,
        db.ptInputOutput,
        db.ptResult:
          begin
            if ClientParams.FindParam(TStoredProc(Aquery).
              Params[i].Name) = nil then
              ClientParams.CreateParam(TStoredProc(AQuery).
                Params[i].DataType,
  TStoredProc(AQuery).Params[i].Name,
  ParamType(ord(TStoredProc(AQuery).Params[i].
                ParamType)));
            ClientParams.ParamByName(TStoredProc(AQuery).
              Params[i].Name).Value :=
              TStoredProc(AQuery).Params[i].Value;
          end;
      end;
```

```
    end;
    except
      LogException(ClientSocket, Exception(ExceptObject).Message,
        True);
    end;
  end;
```

1.8.2.4.3.42 TAstaServerSocket.OnSubmitSQL

TAstaServerSocket

OnSubmitSQL has been deprecated in ASTA 3 and replaced by OnSubmitSQLParams.

Declaration

```
property OnSubmitSQL: TAstaServerExecSQLEvent;
type TAstaServerExecSQLEvent = procedure(Sender: TObject; AQuery:
  TComponent; DataBaseStr, SQLString: string;
  var TheResult: Boolean; var Msg: string) of object;
```

Description

This is the event coded that allows Select statements to work for ASTA clients. Here is an example from the AstaBDEServer. Depending on the threading model in effect, the AQuery parameter will be supplied by the ThreadedDBSupplyQueryEvent. Since ASTA suports any type of component for use with ASTA servers, the example below type casts AQuery as a BDE TQuery since this example was from the AstaBDEServer.

```
  procedure TForm1.AstaServerSocket1SubmitSQL(Sender: TObject;
    ClientSocket: TCustomWinSocket; AQuery: TDataSet; DataBaseStr,
    SQLString: string; Options: TAstaSelectSQLOptionSet);
  begin
    try
      //This is the key to make your server work with ASTA.
      //Just pass on the SQL from the client!
      if TQuery(AQuery).Active then TQuery(AQuery).Close;
      TQuery(AQuery).SQL.Clear;
      TQuery(AQuery).SQL.Add(SQLString);
      TQuery(AQuery).Open;
    except
      LogException(ClientSocket, Exception(exceptObject).Message, True);
    end;
  end;
```

1.8.2.4.3.43 TAstaServerSocket.OnSubmitSQLParams

TAstaServerSocket

Declaration

```
property OnSubmitSQLParams: TAstaParamQueryEvent;
type TAstaParamQueryEvent = procedure(Sender: TObject; ClientSocket:
  TCustomWinSocket; AQuery: TDataSet; const DataBaseStr, SQLString: string;
  SQLParams: TAstaParamList) of object;
```

Description

This is the event that when coded allows Select statements to work for ASTA clients and replaces the Asta 2 OnSubmitSQL event which is now deprecated. In ASTA 3 all Params are no longer expanded on the client but instead passed through to the server. In order to support this the TAstaServerSocket.OnServerQuerySetParams must be coded.

Below is an example from AstaADOSupplement.pas that implements the OnSubmitSQLParams event. The AstaServerSocket.ServerSetSQLParams internally calls the OnServerQuerySetParams event.

```
procedure TAstaADOPlugin.ADOPlugSubmitSQLParams(Sender: TObject;
  ClientSocket: TCustomWinSocket; AQuery: TDataSet; const DataBaseStr,
  SQLString: string; SQLParams: TAstaParamList);
begin
  try
    if AQuery.Active then AQuery.Close;
    TADOQuery(AQuery).SQL.Clear;
    TADOQuery(AQuery).SQL.Add(SQLString);
    ServerSocket.ServerSetSQLParams(Sender, ClientSocket, AQuery,
  SQLParams);
    TADOQuery(AQuery).Open;
  except
    LogException(ClientSocket, Exception(exceptObject).Message, True);
  end;
end;
```

1.8.2.4.3.44 TAstaServerSocket.OnTransactionBegin

TAstaServerSocket

Declaration

```
property OnTransactionsBegin: TAstaBeginTransactionEvent;
type TAstaBeginTransactionEvent = procedure(Sender: TObject; ClientSocket:
TCustomWinSocket; Session: TComponent;
TransActionName: string) of object;
```

Description

This is the event coded that allows a transaction to be started on ASTA servers

Here is an example from the AstaBDEServer. Depending on the threading model in effect, the Session parameter will be supplied by the ThreadedDBSupplyQueryEvent. In this case the ThreadedDBSupplyQuery event receives a ttTransactionStart and returns a TDataBase which the BDE uses to start a transaction with the StartTransaction method. Since ASTA suports any type of component for use with ASTA servers, the example below type casts the Session parameter as a BDE TDataBase since this example was from the AstaBDEServer.

```
procedure TForm1.AstaServerSocket1TransactionBegin(Sender: TObject;
ClientSocket: TCustomWinSocket; Component; TransActionName: string);
begin
  try
    with Session as TDataBase do begin
      if not IsSQLBased then TransIsolation := tiDirtyRead;
      if not InTransaction then StartTransaction;
    end;
  except
    LogException(ClientSocket, Exception(ExceptObject).Message, True);
```

```
end;
end;
```

1.8.2.4.3.45 TAstaServerSocket.OnTransactionEnd

TAstaServerSocket

Declaration

```
property OnTransactionEnd: TAstaEndTransactionEvent;
type TAstaEndTransactionEvent = procedure(Sender: TObject;
  ClientSocket: TCustomWinSocket; Session: TComponent;
  Success: Boolean) of object;
```

Description

This is the event when coded that ends a transaction on ASTA servers.

Coding ASTA Servers

1.8.2.4.3.46 TAstaServerSocket.OnTrigger

TAstaServerSocket

Declaration

```
property OnFireTrigger: TTriggerEvent;
type TTriggerEvent = procedure(Sender: TObject; SQLString: string; U:
  TUserRecord; SQLParser: TAstaSQLParser; P: TAstaParamList; AQuery:
  TComponent) of object;
```

Description

All SQL executed within a transaction on the server from a client initiatied ApplyUpdates call will also flow through the OnTrigger event. The TAstaSQLParser is available to deconstruct the SQL so that you can access the UpdateTableName and Fields.

Example

```
procedure TForm1.AstaServerSocket1Trigger(Sender:TObject; SQLString:
  string; U: UserRecord; SQLParser: TAstaSQLParser; P:TAstaParamList;
  AQuery: TComponent);
begin
  case SQLParser.SQLStatementType of
    stUpdate, stInsert, stDelete:
      AstaServerSocket1.RecordServerActivity(U.FClientSocket,
  SQLString);
  end;
end;
```

1.8.2.4.3.47 TAstaServerSocket.OnTriggerDefaultValue

TAstaServerSocket

Declaration

```
property OnTriggerDefaultValue: TTriggerDefaultValueEvent;
type TTriggerDefaultValueEvent = procedure(Sender:TObject; U: UserRecord;
  SQLString, Tablename, FieldName: string; P: TAstaParamList; AQuery:
  TComponent) of object;
```

Description
ASTA triggers can be defined by calling the AstaClientSocket.RegisterTrigger event so that a parameterized query is created when an ApplyUpdates call is made but the Param will recieve it's value from the server. The value for the field needs to be supplied from the client application before the login process or on the server within the login event. All SQL executed within a transaction on the server from a client initiatied ApplyUpdates call will also flow through the OnTrigger event.

Example

```
procedure TForm1.AstaServerSocket1TriggerDefaultValue(Sender: TObject;
  U: UserRecord; SQLString, Tablename, FieldName: string;
  P: TAstaParamList);
begin
  if (CompareText(tablename,'Customer') = 0) and
    (CompareText(fieldname, 'USERID') = 0) then
  begin
    Logit(' server trigger ' + SQLString);
    p.FastAdd('UserID',
      u.FParamList.ParamByName('UserID').AsInteger);
  end;
end;
```

1.8.2.4.3.48 TAstaServerSocket.OnUserEncryption

TAstaServerSocket

Declaration

```
property OnUserEncryption: TAstaServerEncryptionEvent;
type TAstaServerEncryptionEvent = procedure(Sender: TObject; TUserRecord:
  TUserRecord; EncryptIt: Boolean; var TheData: string) of object;
```

Description
This allows you to add different encryption for each client.

1.8.2.4.3.49 TAstaServerSocket.OnUserListChange

TAstaServerSocket

Declaration

```
property OnUserListChange: TUserChangeEvent;
type TUserChangeEvent = procedure(Sender: TObject; Socket:
  TCustomWinSocket; AState: TUserRecordState) of object;
```

Description
This event fires whenever a client is added or removed from the UserList. ClientSockets are not available in the UserList on the OnClientConnect event. If you want to send a message to a client socket from the OnClientConnect event, use the EarlyRegisterSocket method.

The OnUserListChange event used to be passed an IsNew parameter of type Boolean, but this did not accurately reflect what happened on the server. A remote client first connects to a server, goes through a login process, and then can be disconnected. The new parameter of AState: TUserRecordState more accurately reflects the action of the

ClientSocket. TUserRecordState is defined in AstaTypes.pas

Here is an example of the way servers update the UserDataSet that is on the Asta demo servers. Note: Clients need to come in with login info for this to accurately reflect all states.

```
procedure TForm1.UpdateUserDataSet(Address: string; AState:
  TUserRecordState);
begin
   with UserDataSet do begin
     Append;
     FieldbyName('Address').AsString := Address;
     case Astate of
       tuConnect:
         FieldbyName('Activity').AsString := 'Connected';
       tuDisconnect:
         FieldbyName('Activity').AsString := 'Disconnected';
       tuLogin:
         FieldbyName('Activity').AsString := 'Login';
     end;
     FieldByName('Date').AsDATeTime := Now;
     Post;
  end;
end;

procedure TForm1.AstaServerSocket1UserListChange(Sender: TObject;
  Socket: TCustomWinSocket; AState: TUserRecordState);
begin
  UpdateUserDataSet(Socket.RemoteAddress, Astate);
end;
```

1.8.2.4.3.50 TAstaServerSocket.OnUserRecordStateChange

Declaration

Description

1.8.2.4.3.51 TAstaServerSocket.PersistentFieldTranslateEvent

TAstaServerSocket

Declaration

```
property PersistentFieldTranslateEvent: TAstaFieldTranslate;
type TAstaFieldTranslate = procedure (Sender:TObject; DataSet: TDataSet;
  Field: TField; var NewField: TField) of object;
```

Description

When coding ASTA servers using Providers and ServerMethods persistent Fields can be defined on the server changing things like display labels and displaywidth. This is one of the benefits of server side programming allowing these changes to be coded once and then used on client apps with no changes needed to remote clients.

ASTA supports this by default in streaming down all persistent Fields properties from ServerMethods and TAstaProviders unless the SendFieldProperties is set to false to short

circuit this feature.

There are problems however when the Delphi 3rd Party Database components on the server user Fields Types that don't exist on the client. An example of this, which has bitten many an ASTA user is the TIBStringField used in IBExpress instead of a TStringField. Of course remote clients know nothing of "IB" specific classes as remote ASTA thin clients are world citizens with no alligiance to any specific DataBase.

To allow the best of allow worlds the AstaServerSocket now has a new event named OnPersisentFieldTranslate that is of type TAstaFieldTranslate.

Here is an example from a TAstaIBExpress server to allow any TIBStringFields to be send back to remote clients as normal TString fields.

```
procedure TfrmMain.IBExpressPersisentFieldTranslate(Sender: TObject;
  DataSet: TDataSet; Field: TField; var NewField: TField);
begin
  //no need to dispose if it ASTA will do it internally
  if not (Field is TIBStringField) then Exit;
  NewField := TStringField.Create(DataSet);
  //NewField.Dataset := DataSet; don't set the dataset or it will puke
  NewField.FieldName := Field.FieldName;
  NewField.DisplayLabel := Field.DisplayLabel;
  NewField.Size := Field.Size;
  NewField.Index := Field.Index;
end;
```

1.8.2.4.3.52 TAstaServerSocket.ThreadedDBSupplyQuery

TAstaServerSocket

Declaration

```
property ThreadDBSupplyQuery: TAstaServerSupplyQueryEvent;
type TAstaServerSupplyQueryEvent = procedure(Sender: TObject; ClientSocket:
  TCustomWinSocket; DBAction: TThreadDbAction; DataBaseStr: String;
  CreateNew: Boolean; var AQuery: Component; SQLOptions:
  TAstaSelectSQLOptionSet) of object;
```

Description

When an ASTAclient is connected to an ASTA server and it needs to do a Select or an Exec, or to execute a stored procedure, this routine will be called to create a new component to be used in the client process. Under the BDE, a TQuery is used for Selects and Execs and a TStoredProc is used for stored procedures. Under the ODBCExpress implementation, a TOEDataSet is created for both. The DBAction indicates the type of process that the component will be used for.

The ThreadedDBSupplyQueryEvent sets the AQuery parameter. This allows you to create a single component like a TQuery for a specific action, or to use a component that was created in a data module. In the AstaBDEServer, TQueries or TStoredProcs are created as needed. Since individual components are created on the fly, they must be disposed of by the ASTA server when they have done their tasks. In this case, make sure the DisposeofQueriesForThreads property is set to True. This will tell the ASTA server to dispose of these components. In the AstaODBCExpress server example, a complete data

module is created for each thread. In this case, the DisposeOfQueriesForThreads would be set to false, as any components that are part of a data module will be disposed of when the DataModule is destroyed.

Note: When ASTA servers utilize the tmPersistentSession Model, and packet fetch requests are received from AstaClientDataSets, components must be created on the fly and not used from data modules. In this case, the incoming TThreadDBAction will be set to ttPacketSelect. See the AstaODBCExpress server for an example of this. Since the component must be kept open to be able to return further packet requests, it must be created on the fly.

When running threaded the Sender parameter will come in as a TAstaThread otherwise in SingleThreaded mode it will be the AstaServerSocket itself.

Example

```
procedure TForm1.AstaServerSocket1ThreadedDBSupplyQuery(Sender: TObject;
ClientSocket: TCustomWinSocket; DBAction: TThreadDbAction; DataBaseStr:
String; CreateNew: Boolean; var AQuery: TComponent; SQLOptions:
TAstaSelectSQLOptionSet);
begin
  try
    if CreateNew then begin
      case DBAction of
        ttStoredProc:
          begin
            AQuery := TStoredProc.Create(nil);
            with AQuery as TStoredProc do begin
              SessionName := TDataBase(TAstaThread(Sender).
                Session).SessionName;
              DataBaseName := TDataBase(TAstaThread(Sender).
                Session).DataBaseName;
            end;
          end;
        ttTransactionStart:
          AQuery := TDataBase(TAstaThread(Sender).Session);
      else
        AQuery := TQuery.Create(nil);
        with AQuery as TQuery do begin
          SessionName := TDataBase(TAstaThread(Sender).Session).
            Session.SessionName;
          DataBaseName := TDataBase(TAstaThread(Sender).Session).
            DataBaseName;
        end
      end;
    end else  //create new
      case DBAction of
        ttStoredProc: AQuery := StoredProc1;
        ttTransactionStart: AQuery := MainDataBase;
      else
        AQuery := Query;
      end;
  except
    LogException(ClientSocket, Exception(exceptObject).Message, True);
  end;
end;
```

1.8.2.4.3.53 TAstaServerSocket.ThreadedDBSupplySession

TAstaServerSocket

Declaration

```
property ThreadDBSupplySession: TAstaSessionCreateEvent;
type TAstaSessionCreateEvent = procedure(Sender: TObject;
  ClientSocket: TCustomWinSocket; DataBaseString: string;
  var ASession: TComponent) of object;
```

Description

When Threading ASTA servers, this event must be coded so that ASTA can create whatever database component is needed to be able to run independently in a thread. This can be a TDatabase/TSession pair under the BDE, or a data module containing any components necessary. The ASession parameter is declared as a TComponent to make this as flexible as is necessary to handle any Delphi database component.

Example

```
procedure TForm1.AstaServerSocket1ThreadedDBSupplySession(
  Sender: TObject; ClientSocket: TCustomWinSocket;
  DataBaseString: string; var ASession: TComponent);
var
  Sess: TSession;
begin
  try
    Sess := TSession.Create(nil); //Asta will dispose of it if
      //CreateSessionForDBThreads was called with true
    ASession := TDataBase.Create(Sess);
    with ASession as TDataBAse do begin
      DataBaseName := 'db' + IntToStr(AstaServerSocket1.
        SessionList.Count);
      Name := 'DataBase' + IntToStr(AstaServerSocket1.
        SessionList.Count);
      AliasName := MainDataBase.AliasName;
      LoginPrompt := False;
      Params.Assign(MainDatabase.Params);
      TransIsolation := MainDataBase.TransIsolation;
      Connected := True;
    end;
  except
    LogException(ClientSocket, Exception(exceptObject).Message, True);
  end;
end;
```

1.8.2.5 TAstaSessionList

Properties : Methods : Events

Unit

AstaSessionList

Applies to

TAstaServerSocket

Declaration

```
type TAstaSessionList = class(TObject);
```

Description
Used internally by ASTA servers to manage database sessions when threaded. In the Pooled Threading Model at server startup the AstaServerSocket calls ThreadedDBSupplySession DatabaseSessions number of times to populate the AstaServerSocket.SessionList in the CreateSessionsForDBThreads method that should be called in the FormCreate of any AstaServer.

An example of how the AstaSessionList could be used is if you want to get a DatabaseSession under the Persistent Session threading model for an ASTA messaging call. You would call AstaServerSocket.SessionList. GetSessionFromSocket(ClientSocket, '', nil);

See Threading ASTA Servers for a complete discussion.

1.8.2.5.1 Properties

Enter topic text here.

1.8.2.5.1.1 TAstaSessionList.FreeSessions

TAstaSessionList

Declaration
```
property FreeSessions: Boolean;
```

Description
Determines whether or not ASTA will free up the Sessions created automatically. It is passed in when the AstaServerSocket.CreateSessionsForDbThreads(FreeSessions: Boolean) method is called. Typically True should be used in this call.

1.8.2.5.1.2 TAstaSessionList.MaximumAsyncSessions

TAstaSessionList

Declaration
```
property MaximumAsyncSessions: Integer;
```

Description

1.8.2.5.1.3 TAstaSessionList.MaximumSessions

TAstaSessionList

Declaration
```
property MaximumSessions: Integer;
```

Description
The session pool expands on demand if a session is not available. The MaximumSessions property will not allow the pool to expand beyond MaximumSessions. Rememember that 8 sessions could easily be handling 50-100 users or more depending on the size and frequency of client requests.

If MaxiumSessions is reached, the thread will be created and added to a ThreadQueue list. When a running process is finished, the thread will be popped from the pool and executed using the newly freed up session.

1.8.2.5.1.4 TAstaSessionList.StartupSessions

TAstaSessionList

Declaration
```
property StartupSessions: Integer;
```

Description
From the AstaServerSocket.DataBaseSessions property. The initial size of the thread pool.

1.8.2.5.2 Methods

Enter topic text here.

1.8.2.5.2.1 TAstaSessionList.AddASession

TAstaSessionList

procedure AddASession(Session: TComponent);

1.8.2.5.2.2 TAstaSessionList.AddASessionSocket

TAstaSessionList

1.8.2.5.2.3 TAstaSessionList.AddASocketwithSession

TAstaSessionList

procedure AddASocketwithSession(Socket: TObject; Session: Tcomponent);

1.8.2.5.2.4 TAstaSessionList.AddDataModuleToSession

TAstaSessionList
procedure AddDataModuleToSession(Session, NewDM: TComponent);

1.8.2.5.2.5 TAstaSessionList.AddPersistentThread

procedure AddPersistentThread(T:TObject);

1.8.2.5.2.6 TAstaSessionList.AddSocketToSession

procedure AddSocketToSession(Index: Integer; S, AT: TObject);

1.8.2.5.2.7 TAstaSessionList.FreeAQuery

procedure FreeAQuery(TheSocket: TObject; Queryid, Proxyid: Integer);

1.8.2.5.2.8 TAstaSessionList.FreeProxyDataModule

procedure FreeProxyDataModule(Socket: Tobject; DMName: string; Proxyid: Integer);

1.8.2.5.2.9 TAstaSessionList.GetActionItemDataModule

function GetActionItemDataModule(ClientSocket: TCustomWinSocket; MethodName: string; at: TObject): TPersistent;

1.8.2.5.2.10 TAstaSessionList.GetActionItemFromDataModule

function GetActionItemFromDataModule(DM: TComponent; Method: string; at: TObject): TPersistent;

1.8.2.5.2.11 TAstaSessionList.GetActionItemFromMethodName

function GetActionItemFromMethodName(ClientSocket: TCustomWinSocket; Method: string; at: TObject): TPersistent;

1.8.2.5.2.12 TAstaSessionList.GetAvailableSession

function GetAvailableSession(Socket: TObject; DataBaseString: string; At: TObject): TComponent;

1.8.2.5.2.13 TAstaSessionList.GetDataModuleListFromSession

function GetDataModuleListFromSession(Session: TComponent): TDataModuleList;

1.8.2.5.2.14 TAstaSessionList.GetDataSetFromProxyName

function GetDataSetFromProxyName(Socket: TObject; DSName, ProxyName: string; Proxyid: Integer): TComponent;

1.8.2.5.2.15 TAstaSessionList.GetPersistentSessionFromSocket

function GetPersistentSessionFromSocket(Socket, At: TObject; DataBaseString: string): TComponent;

1.8.2.5.2.16 TAstaSessionList.GetProviderFromDataModule

function GetProviderFromDataModule(ClientSocket: TCustomWinSocket; ProviderName, DataModule: string; Proxyid: Integer; at: TObject): TComponent;

1.8.2.5.2.17 TAstaSessionList.GetProviderFromSession

function GetProviderFromSession(ClientSocket: TCustomWinSocket; ProviderName: string; At: TObject): TComponent;

1.8.2.5.2.18 TAstaSessionList.GetQueryFromASocket

function GetQueryFromASocket(Socket: TObject; Queryid: Integer): TComponent;

1.8.2.5.2.19 TAstaSessionList.GetSession

function GetSession(Index: Integer): TComponent;

1.8.2.5.2.20 TAstaSessionList.GetSessionData

function GetSessionData(Index: Integer): PSessionData;

1.8.2.5.2.21 TAstaSessionList.GetSessionDataFromSocket

function GetSessionDataFromSocket(Socket: Tobject; at: TObject): PSessionData;

1.8.2.5.2.22 TAstaSessionList.GetSessionFromSocket

function GetSessionFromSocket(Socket, AT: TObject; DataBaseString: string): TComponent;

1.8.2.5.2.23 TAstaSessionList.GetSmartSession

function GetSmartSession(Socket, At: Tobject;DatabaseString:String): TComponent;

1.8.2.5.2.24 TAstaSessionList.InitSessionData

procedure InitSessionData(s: PSessionData);

1.8.2.5.2.25 TAstaSessionList.InitSessions

procedure InitSessionData(s: PSessionData);

1.8.2.5.2.26 TAstaSessionList.MakeAvailable

procedure MakeAvailable(Session: TComponent);

1.8.2.5.2.27 TAstaSessionList.MaxAsyncSessionExceeded

function MaxAsyncSessionExceeded(Socket: TObject): Boolean;

1.8.2.5.2.28 TAstaSessionList.PersistentCleanUpSession

Procedure PersistentCleanUpSession(Sender:TObject);

1.8.2.5.2.29 TAstaSessionList.ProxyDataModuleIsRegistered

function ProxyDataModuleIsRegistered(Socket: TObject; DMName: string; Proxyid: Integer): Boolean;

1.8.2.5.2.30 TAstaSessionList.QueuedCount

function QueuedCount:Integer;

1.8.2.5.2.31 TAstaSessionList.QueuedProcessCheck

procedure QueuedProcessCheck(s: PSessionData);

1.8.2.5.2.32 TAstaSessionList.QueuedProcessLaunch

procedure QueuedProcessLaunch(s: PSessionData);

1.8.2.5.2.33 TAstaSessionList.QueueThread

procedure QueueThread(T: TObject);

1.8.2.5.2.34 TAstaSessionList.RunningPersistentSession

Function RunningPersistentsession(S:PSessionData):Boolean;

1.8.2.5.2.35 TAstaSessionList.SessionsInUse

Function SessionsInUse:Integer;

1.8.2.5.2.36 TAstaSessionList.SetUseFlag

procedure SetUseFlag(Index: Integer; Value: Boolean);

1.8.2.5.2.37 TAstaSessionList.ShrinkSessionList

procedure ShrinkSessionList(newSize:integer);

1.8.2.5.2.38 TAstaSessionList.SocketDisposeSession

procedure SocketDisposeSession(Socket: TObject; PersistentSession: Boolean);

1.8.2.5.2.39 TAstaSessionList.SocketRegisterProxyDataModule

procedure SocketRegisterProxyDataModule(Socket: Tobject; OriginalDMName: string; DM: TComponent; ProxyId: Integer);

1.8.2.5.2.40 TAstaSessionList.SocketValidateList

procedure SocketValidateList(ServerSocket:TObject);

1.8.2.5.2.41 TAstaSessionList.StoreQueryForPacket

function StoreQueryForPacket(D: TDataSet; TheSocket: TObject; PlaceHolder, DataSetMessageid: Integer; ServerObject: TObject; UsedOnServer: Boolean): Integer;

1.8.2.5.2.42 TAstaSessionList.UpdateAsyncList

procedure UpdateAsyncList(Socket: TObject; Amt: Integer; UseCriticalSection: Boolean);

1.8.2.5.3 Events

Enter topic text here.

1.8.2.5.3.1 TAstaSessionList.OnAddDataModule

property OnAddDataModule: TAddDataModuleEvent;

TAddDataModuleEvent = procedure(Sender: TObject; Session: TComponent; DMList: TList) of object;

1.8.2.5.3.2 TAstaSessionList.OnPacketQueryDispose

property OnPacketQueryDispose: TPacketQueryFreeEvent;
TPacketQueryFreeEvent = procedure(Sender: TObject; ClientSocket: TCustomWinSocket; AQuery: TDataSet; var DisposeofIt: Boolean) of object;

1.8.2.5.3.3 TAstaSessionList.CreateASession

property CreateASession: TAstaSessionCreateEvent;

1.8.2.6 TAstaThread

Properties

Unit
AstaThread

Declaration
```
type TAstaThread = class(TThread);
```

Description
ASTA was designed so that developers would not have to worry about implementing any threading. ASTA servers are threaded at the ASTA component level.

In some of the AstaServerSocket events the Sender (TObject) parameter is a TAstaThread, and there may be times when you need to typecast this parameter to use some of the properties of the TAstaThread. See TAstaUtilityEvent.

1.8.2.6.1 Properties

TAstaThread

ServerSocket
Session
TheQuery
TheSocket
ThreadDBAction
ThreadedClient

1.8.2.6.1.1 TAstaThread.Data

TAstaThread

Declaration
```
property Data: Pointer;
```

Description

1.8.2.6.1.2 TAstaThread.DataBase

TAstaThread

Declaration
```
property DataBase: TComponent;
```

Description

1.8.2.6.1.3 TAstaThread.FreeQuery

TAstaThread

Declaration
```
property FreeQuery: Boolean;
```

Description

1.8.2.6.1.4 TAstaThread.FreeSessions

TAstaThread

Declaration
```
property FreeSessions: Boolean;
```

Description

1.8.2.6.1.5 TAstaThread.Params

TAstaThread

Declaration
```
property Params: TAstaParamList;
```

Description

1.8.2.6.1.6 TAstaThread.ServerSocket

TAstaThread

Declaration
```
property ServerSocket: TServerSocket;
```

Description
This is a pointer back to the AstaServerSocket so that the main process AstaServerSocket is not accessed from the thread. It is safer to typecast this property.

1.8.2.6.1.7 TAstaThread.Session

TAstaThread

Declaration
```
property Session: TComponent;
```

Description
This is the component created in Pooled or Persistent Threading, typically a DataModule. It will be created on the ASTA server in the ThreadedDBSupplyQueryEvent and OnClientDBLogin events.

1.8.2.6.1.8 TAstaThread.SessionListHolder

TAstaThread

Declaration
```
property SessionListHolder: Pointer;
```

Description

1.8.2.6.1.9 TAstaThread.TheQuery

TAstaThread

Declaration
```
property TheQuery: TComponent;
```

Description
This is the component used for the database threading activity and the one called from AstaServerSocket.ThreadedDBSupplyQuery.

1.8.2.6.1.10 TAstaThread.TheSocket

TAstaThread

Declaration
```
property TheSocket: TCustomWinSocket;
```

Description
This is the client socket which is useful if you want to use ASTA messaging calls.

1.8.2.6.1.11 TAstaThread.ThreadDBAction

TAstaThread

Declaration
```
property ThreadDBAction: TThreadDBAction;
```

Description
This is the type of activity the thread is going to perform and is passed to ThreadedDBSupplyQuery to return the appropriate component for the thread process.

1.8.2.6.1.12 TAstaThread.ThreadedClient

TAstaThread

Declaration
```
property ThreadedClient: TUserRecord;
```

Description
When a TAstaThread is created it makes a copy of the UserRecord for use within the thread.

See ThreadedUtilityEvent for examples of how this is used.

1.8.2.6.2 Methods

TAstaThread

1.8.2.6.2.1 TAstaThread.AddToPersistentList

Declaration
procedure AddToPersistentList;

Description

1.8.2.6.2.2 TAstaThread.HasTerminated

Declaration
function HasTerminated: Boolean;

Description

1.8.2.6.2.3 TAstaThread.MakeSessionAvailable

Declaration
procedure MakeSessionAvailable;

Description

1.8.2.6.2.4 TAstaThread.PerformanceString

Declaration
function PerformanceString(S: string): string;

Description

1.8.2.6.2.5 TAstaThread.ResetQueryForThread

Declaration
procedure ResetQueryForThread;

Description

1.8.2.6.2.6 TAstaThread.ResetServerMethodForThread

Declaration
procedure ResetServerMethodForThread;

Description

1.8.2.6.2.7 TAstaThread.Run

Declaration
procedure Run;

Description

1.8.2.6.2.8 TAstaThread.SetUserRecord

Declaration
procedure SetUserRecord(U: TUserRecord);

Description

1.8.2.6.3 Events

TAstaThread

1.8.2.6.3.1 TAstaThread.CustomDBUtilityEvent

Declaration
property CustomDBUtilityEvent: TAstaDBUtilityEvent;

Description

1.8.2.6.3.2 TAstaThread.CustomUtilEvent

Declaration
property CustomUtilEvent: TAstaUtilityEvent;

Description

1.8.2.6.3.3 TAstaThread.OnAfterExecute

TAstaThread

Declaration

```
property OnAfterExecute: TTreadDBEvent;

TTreadDBEvent = procedure(Sender: TObject; Session, Query: TComponent) of
  object;
```

Description

1.8.2.6.3.4 TAstaThread.OnExecute

Declaration
property OnExecute: TTreadDBEvent;

Description

1.8.2.7 TUserRecord

Properties : Methods

Unit
AstaTypes

Declaration
```
type TUserRecord = class(TObject);
```

Description

```
  procedure Login(const UName, AppName: string);
  procedure Log(Msg: string);

  function PersistentSession: Boolean;
  property DatabaseSession: TComponent read FDatabaseSession write FDatabaseSession;
  property OnStateChange: TUserRecordEvent read FOnStateChange write FOnStateChange;
  function IsValid: Boolean;

  function IsPalm: Boolean;
  function IsRemotePda: Boolean;
  function isStateless: Boolean;
  property ClientSocket: TCustomWinSocket read fClientSocket write fClientSocket;
  property ParamList: TAstaParamList read FParamList write FParamList;
  property RemoteUser: TRemoteUser read fRemoteUser write fRemoteUser;
  property MessageStack: TAstaServerMessageStack read fMessageStack;
  property StringLine: TAstaStringLine read fStringLine;
  property UserName: string read FUserName write FUserName;
  property AppName: string read FAppName write FAppName;
  property AppVersion: string read FAppVersion write FAppVersion;
  property ConnectTime: TDateTime read FConnectTime write FConnectTime;
  property SupportsCompression: Boolean read FSupportsCompression write
FSupportsCompression;
  property SupportsEncryption: Boolean read FSupportsEncryption write FSupportsEncryption;
  property StatelessMessagingThreadLaunched: Boolean read GetStatelessMessageThreaded;
  property UserList: TObject read FUserList write FUserList;
{$IFDEF AstaHttp}
  property Protocol: TAstaProtocol read FProtocol write FProtocol;
{$ENDIF}
  property SelfCrypted: Boolean read FEncrypt;
  property UserKey: string read FUserKey write FUserKey;
  property EncryptKey: string read AuthData write AuthData;
  property MaxAsyncPool: Integer read FMaxAsyncPool write FMaxAsyncPool;
  property InDBName: string read FInDBName write FInDBName;
  property InDBStream: TStream read FInDBStream write FInDBStream;
```

1.8.2.7.1 Properties

1.8.2.7.1.1 TUserRecord.MaxResponseSize

property MaxResponseSize:Integer read FMaxResponseSize write FMaxResponseSize;

1.8.2.7.2 Methods

1.8.2.7.2.1 TUserRecord.PersistentSession

Enter topic text here.

Function PersistentSession:Boolean;

1.8.2.8 TAsta2MetaData

Properties : Methods : Events

Unit
Asta2MetaData

Declaration
```
type TAsta2MetaData = class(TComponent);
```

Description
Because Asta supports so many different 3rd party database components, many of the Asta servers have been coded to return different field names for the metadata. For example, one server returns FieldSize and another server returns Size. If a client application makes use of the field names returned by the metadata events, it is very difficult to use that same client against different Asta servers.
The Asta2Metadata component standardizes the field names returned for all metadata events. The component also ensures that even if a server does not implement certain metadata calls, it always returns a valid dataset. In the case where the server does not implement a certain metadata call, the dataset will be empty. But at least the client application will not fail because it can not find any metadata fields.

The component implements a standard set of fieldnames to return.

MetaDataRequest	FieldName	DataType	Size
mdOtherMetaData	Memo	ftMemo	0
mdTables	TableName	ftString	30
	OwnerName	ftString	30
	Description	ftString	30
mdIndexes	IndexName	ftString	30
mdFields	FieldName	ftString	30
	FieldType	ftString	30
	FieldSize	ftInteger	0
mdViews	ViewName	ftString	30
	OwnerName	ftString	30
	Description	ftString	30
mdStoredProcs	SProcName	ftString	30
	OwnerName	ftString	30
	Description	ftString	30
mdForeignKeys	FieldName	ftString	30
	TableName	ftString	30
	IndexName	ftString	30
mdSystemTables	TableName	ftString	30
	OwnerName	ftString	30
	Description	ftString	30
mdPrimeKeys	FieldName	ftString	30
mdStoredProcColumns	ColumnName	ftString	30
	ColumnType	ftString	30
	ColumnSize	ftString	30
mdVCLFields	FieldName	ftString	30
	FieldType	ftString	0
	FieldSize	ftString	0
mdDBMSName	AstaServer	ftString	25
	Database	ftString	40
	Info	ftString	40
	Server EXE Name	ftString	40
	UserName	ftString	35
	Password	ftString	35
	DataBaseFileName	ftMemo	0

If any other fields or more fields are required, they can be added in the individual events.

The idea of this component is that it gives you a dataset in the event code, that already contains the dataset fields. All you have to do is to populate it with your data values.

Usage:

The AstaServerSocket's OnFetchMetadata event must be coded to call the GetMetadata method of the component:

```
try
AstaServerSocket1.MetaDataSet:=AstaMetaData.GetMetaData(Sender,ClientSocket,
MetaRequest, DatabaseStr, Arg1);
except
LogException(ClientSocket, 'Stored MetaData Error: ' +
EDataBaseError(ExceptObject).Message, True);
end;
```

Each event must be individually coded. Below are some examples from the IBX server:

```
/* The FibInfo dataset below is an Asta component that just returns the metadata values.
This event populates the MetaDataSet dataset, which will be returned to the client
application */

procedure TfrmMain.AstaMetaDataFields(Sender: TObject;ClientSocket:
TCustomWinSocket; MetaDataRequest: TAstaMetaData;
MetaDataSet: TAstaDataSet; DatabaseName, TableName: String);
begin
LogIt('Fields Info for ' + TableName);
FIbInfo.GetMetaData(MetaDataRequest, DatabaseName, TableName);
FIbInfo.DataSet.First;
while not FIbInfo.DataSet.Eof do
begin

MetaDataSet.AppendRecord([Trim(FIbInfo.DataSet.Fields[0].AsString),

Ord(FIbInfo.IBTypeToVCLFieldType(FIbInfo.DataSet.Fields[1].AsInteger))
,
FIbInfo.DataSet.Fields[2].AsInteger]);
FIbInfo.DataSet.Next;
end;
end;
```

If you need more or other fields in the metadata dataset, you have to close the MetaDataSet, nuke the fields, add new fields and open the dataset again. The dataset MUST be opened at the end of the event.The example below is from the AstaCTLibServer:

```
procedure Tf_AstaCTLibServer.AstaMetaDataFields(Sender: TObject;ClientSocket:
TCustomWinSocket; MetaDataRequest: TAstaMetaData;MetaDataSet: TAstaDataSet;
DatabaseName, TableName: String);var i :Integer;
begin
LogIt('Fields Info for ' + TableName);
with TAstaDataModule(AstaMetaData.DataModule) do
begin
CTInfo.SelectDb:=DatabaseName;
if TableName = '' then exit;
CTInfo.GetBasicColumnsInfo(trim(TableName));
try
```

```
with MetaDataSet do
begin
Close;
NukeAllFieldInfo;
FastFieldDefine('FieldName', ftString, 30);
FastFieldDefine('FieldType', ftInteger, 0);
FastFieldDefine('FieldSize', ftInteger, 0);
FastFieldDefine('SybaseType', ftString, 30);
FastFieldDefine('FieldPrecision', ftInteger, 0);
FastFieldDefine('FieldScale', ftInteger, 0);
Open;
Empty;
end;

for i:=1 to CTInfo.NumCols do
begin
MetaDataSet.AppendRecord([CTInfo.Heading[i],

ord(SybTypeToVCLFieldType(CTInfo.ColSybType[i])),
CTInfo.ColLen[i],
CTInfo.ColSybType[i],
CTInfo.ColPrecision[i],
CTInfo.ColScale[i]
]);
end;
finally
CTInfo.FreeBasicColumns;
end;
end;
end;
```

Using the Asta2Metadata component, you do not need to code ANY of the metadata events. The metadata will then only return an empty dataset.

1.8.2.8.1 Properties

TAsta2MetaData

AliasFileName
AstaServerSocket
DataModule
MetaData
MetaDataList

1.8.2.8.1.1 TAsta2MetaData.MetaData

TAsta2MetaData

Declaration
```
property MetaData[MetaDataRequest: TAstaMetaData]: TMetaData;
```

Description

1.8.2.8.1.2 TAsta2MetaData.MetaDataList

TAsta2MetaData

Declaration
```
property MetaDataList: TList;
```

Description

1.8.2.8.2 Methods

TAsta2MetaData

AdjustAlias
InitDataSet
GetMetaData

1.8.2.8.2.1 TAsta2MetaData.AdjustAlias

TAsta2MetaData

Declaration
```
procedure AdjustAlias(ClientSocket: TCustomWinSocket; Database: string;
  Query: TComponent; DBAction: TThreadDbAction);
```

Description

1.8.2.8.2.2 TAsta2MetaData.InitDataSet

TAsta2MetaData

Declaration
```
procedure InitDataSet(MetaDataRequest: TAstaMetaData);
```

Description

1.8.2.8.2.3 TAsta2MetaData.GetMetaData

TAsta2MetaData

Declaration
```
function GetMetaData(Sender: TObject; ClientSocket: TCustomWinSocket;
  MetaDataRequest: TAstaMetaData; DatabaseName: string;
  TableName: string): TAstaDataSet;
```

Description
GetMetaData is the main call to fetch the metadata from the server. Supply the type of metadata in MetaDataRequest.

1.8.2.8.3 Events

TAsta2MetaData

AfterMetaData
BeforeMetaData
OnDBMSName
OnFields
OnForeignKeys
OnIndexes
OnOtherMetaData
OnPrimeFields
OnStoredProcColumns
OnStoredProcs
OnSystemTables
OnTables
OnTriggers
OnVCLFields
OnViews

1.8.2.8.3.1 TAsta2MetaData.AfterMetaData

TAsta2MetaData

Declaration
```
property AfterMetaData: TAfterMetaDataEvent;
TAfterMetaDataEvent = procedure(Sender: TObject; MetaDataRequest:
  TAstaMetaData; DatabaseName: string; TableName: string) of object;
```

Description
This event is called directly after GetMetData is called.

1.8.2.8.3.2 TAsta2MetaData.BeforeMetaData

TAsta2MetaData

Declaration
```
property BeforeMetaData: TBeforeMetaDataEvent;
TBeforeMetaDataEvent = procedure(Sender: TObject; MetaDataRequest:
  TAstaMetaData; DatabaseName: string; TableName: string) of object;
```

Description
This event is called directly before GetMetData is called.

1.8.2.8.3.3 TAsta2MetaData.OnDBMSName

TAsta2MetaData

Declaration
```
property OnDBMSName: TOnDBMSNameEvent;
TOnDBMSNameEvent = procedure(Sender: TObject; ClientSocket:
  TCustomWinSocket; MetaDataSet: TAstaDataSet) of object;
```

Description
Used to respond to mdDBMSName. Used when the server supports multiple aliases.

1.8.2.8.3.4 TAsta2MetaData.OnFields

TAsta2MetaData

Declaration
```
property OnFields: TOnFieldsEvent;
TOnFieldsEvent = procedure(Sender: TObject; ClientSocket: TCustomWinSocket;
  MetaDataRequest: TAstaMetaData; MetaDataSet: TAstaDataSet; DatabaseName:
  string; TableName: string) of object;
```

Description
Response to mdFields returns Fieldname, Fieldtype and size info.

1.8.2.8.3.5 TAsta2MetaData.OnForeignKeys

TAsta2MetaData

Declaration
```
property OnForeignKeys: TOnForeignKeysEvent;
TOnForeignKeysEvent = procedure(Sender: TObject; ClientSocket:
  TCustomWinSocket; MetaDataRequest: TAstaMetaData; MetaDataSet:
  TAstaDataSet; DatabaseName: string; TableName: string) of object;
```

Description
Fires when returning foreign key info.

1.8.2.8.3.6 TAsta2MetaData.OnIndexes

TAsta2MetaData

Declaration
```
property OnIndexes: TOnIndexesEvent;
TOnIndexesEvent = procedure(Sender: TObject; ClientSocket:
  TCustomWinSocket; MetaDataRequest: TAstaMetaData; MetaDataSet:
  TAstaDataSet; DatabaseName: string; TableName: string) of object;
```

Description
Fires when returning Index info.

1.8.2.8.3.7 TAsta2MetaData.OnOtherMetaData

TAsta2MetaData

Declaration
```
property OnOtherMetaData: TOnOtherMetaDataEvent;
TOnOtherMetaDataEvent = procedure(Sender: TObject; ClientSocket:
  TCustomWinSocket; MetaDataRequest: TAstaMetaData; MetaDataSet:
  TAstaDataSet; DatabaseName: string; TableName: string) of object;
```

Description
Fires when returning other metadata info.

1.8.2.8.3.8 TAsta2MetaData.OnPrimeKeys

TAsta2MetaData

Declaration

```
property OnPrimeKeys: TOnPrimeKeysEvent;
TOnPrimeKeysEvent = procedure(Sender: TObject; ClientSocket:
  TCustomWinSocket; MetaDataRequest: TAstaMetaData; MetaDataSet:
  TAstaDataSet; DatabaseName: string; TableName: string) of object;
```

Description

Fires when returning primary key key info.

1.8.2.8.3.9 TAsta2MetaData.OnStoredProcColumns

TAsta2MetaData

Declaration

```
property OnStoredProcColumns: TOnStoredProcColumnsEvent;
TOnStoredProcColumnsEvent = procedure(Sender: TObject; ClientSocket:
  TCustomWinSocket; MetaDataRequest: TAstaMetaData; MetaDataSet:
  TAstaDataSet; DatabaseName: string; TableName: string) of object;
```

Description

Fires when returning stored procedure column info.

1.8.2.8.3.10 TAsta2MetaData.OnStoredProcs

TAsta2MetaData

Declaration

```
property OnStoredProcs: TOnStoredProcsEvent;
TOnStoredProcsEvent = procedure(Sender: TObject; ClientSocket:
  TCustomWinSocket; MetaDataRequest: TAstaMetaData; MetaDataSet:
  TAstaDataSet; DatabaseName: string; TableName: string) of object;
```

Description

Fires when returning stored procedure info.

1.8.2.8.3.11 TAsta2MetaData.OnSystemTables

TAsta2MetaData

Declaration

```
property OnSystemTables: TOnSystemTablesEvent;
TOnSystemTablesEvent = procedure(Sender: TObject; ClientSocket:
  TCustomWinSocket; MetaDataRequest: TAstaMetaData; MetaDataSet:
  TAstaDataSet; DatabaseName: string; TableName: string) of object;
```

Description

Fires when returning system table info.

1.8.2.8.3.12 TAsta2MetaData.OnTables

TAsta2MetaData

Declaration

```
property OnTables: TOnTablesEvent;
TOnTablesEvent = procedure(Sender: TObject; ClientSocket: TCustomWinSocket;
    MetaDataRequest: TAstaMetaData; MetaDataSet: TAstaDataSet;
    DatabaseName: string; TableName: string) of object;
```

Description

Fires when returning table info.

1.8.2.8.3.13 TAsta2MetaData.OnTriggers

TAsta2MetaData

Declaration

```
property OnTriggers: TOnTriggersEvent;
TOnTriggersEvent = procedure(Sender: TObject; ClientSocket:
  TCustomWinSocket; MetaDataRequest: TAstaMetaData; MetaDataSet:
  TAstaDataSet; DatabaseName: string; TableName: string) of object;
```

Description

Fires when returning trigger info.

1.8.2.8.3.14 TAsta2MetaData.OnVCLFields

TAsta2MetaData

Declaration

```
property OnVCLFields: TOnVCLFieldsEvent;
TOnVCLFieldsEvent = procedure(Sender: TObject; ClientSocket:
  TCustomWinSocket; MetaDataRequest: TAstaMetaData; MetaDataSet:
  TAstaDataSet; DatabaseName: string; TableName: string) of object;
```

Description

Fires when returning VCL field info.

1.8.2.8.3.15 TAsta2MetaData.OnViews

TAsta2MetaData

Declaration

```
property OnViews: TOnViewsEvent;
TOnViewsEvent = procedure(Sender: TObject; ClientSocket: TCustomWinSocket;
  MetaDataRequest: TAstaMetaData; MetaDataSet: TAstaDataSet; DatabaseName:
  string; TableName: string) of object;
```

Description

Fires when returning View info.

1.8.2.9 TAstaBDEInfoTable

Properties

Unit
AstaBDEInfo

Declaration
```
type TAstaBDEInfoTable = class(TTable)
```

Description
Use TAstaBDEInfoTable to retrieve database metadata information using the BDE. Set the BDEInfo property to the type of information to retrieve and then set the Active property to True.

You can use TAstaBDEInfoTable just like a standard TTable once it is populated with the metadata information that you requested.

1.8.2.9.1 Properties

TAstaBDEInfoTable

BDEInfo

1.8.2.9.1.1 TAstaBDEInfoTable.BDEInfo

TAstaBDEInfoTable

Declaration
```
property BDEInfo: BDEInfoType;
type BDEInfoType = (BDENOInfo, BDEUserInfo, BDEFieldInfo, BDEIndexInfo,
  BDELockList, BDEOpenTables, BDEStoredProcParams, BDEStoredProcedures,
  BDESystemStoredProcedures);
```

Description
Sets the type of info that will be returned.

Value	Meaning
BDENOInfo	Does not return any BDE specific information. Identical to TTable.
BDEUserInfo	Returns the current BDE users accessing the table specified by the TableName property.
BDEFieldInfo	Returns field information for TableName.
BDEIndexInfo	Returns index information for TableName.
BDELockList	BDELockList - Returns current BDE locks information.
BDEOpenTables	Returns a summary of all tables for the current database specified in the DatabaseName property.
BDEStoredProcParams	Returns parameter information for the stored procedure specified in TableName.
BDEStoredProcedures	Returns a summary of all stored procedures for the current database.
BDESystemStoredProcedures	Returns a summary of all system stored procedures for the current database.

1.8.2.10 TAstaSQLGenerator

Properties : Events

Unit
AstaSQLGenerate

Declaration
```
type TAstaSQLGenerator = class(TComponent)
```

Description
The TAstaSQLGenerator can be used on ASTA servers to generate SQL on the server. The TAstaServerSocket has a ServerSQLGenerator property to connect to a TAstaSQLGenerator. It allows server side SQL settings much like the TAstaClientSocket has to be able to customize the SQL that ASTA generates. TAstaProviders use a TAstaSQLGenertor to guide them in the SQL syntax that they generate. If no TAstaSQLGenerator is present, the TAstaProvider will create and release one on the fly. You need only one TAstaSQLGenerator per ASTA server and they should be on the same form as the TAstaServerSocket.

1.8.2.10.1 Properties

TAstaSQLGenerator

DateMaskForSQL
DateTimeMaskForSQL
ServerSQLOptions
SQLDialect
UpdateMode
UpdateSQLSyntax

1.8.2.10.1.1 TAstaSQLGenerator.ServerSQLOptions

TAstaSQLGenerator

Declaration
```
property ServerSQLOptions: TAstaServerSQLOptions;
```

Description
The ServerSQLOptions property of the TAstaSQLGenerator on the server allows for full customization of the SQL that ASTA generates. The ServerSQLOptions set is automatically filled if the UpdateSQLSyntax property is changed.

ASTA clients and servers can be configured a variety of ways to provide the SQL syntax used for your database. For a full discussion see SQL Generation.

1.8.2.10.2 Events

TAstaSQLGenerator

OnCustomParamSyntax
OnCustomSQLSyntax

1.8.2.11 TAstaSQLParser

Properties : Methods

Unit
AstaSQLParser

Declaration
```
type TAstaSQLParser = class(TComponent);
```

Description
There are times when you want to modify your SQL like changing the WHERE clause, ORDER BY, fields, tables, HAVING and GROUP BY clauses. The AstaClientDataSet now has a built in SQL parser that can deconstruct your SQL and allow you to piece it back together.

The AstaSQLParser can take any SQL by setting the SQL Property.

```
SQLParser.SQL.Text := YourSQL.Text;
SQLParser.Deconstruct;
```

When Deconstruct is called the follow properties are populated:

Property	Description
UpdateTable	The table used in the UPDATE statement
DeleteTable	The table used in the DELETE statement
InsertTable	The table used in the INSERT statement
Tables	List of the tables in the SQL statement
Correlations	The list of table correlations. In the SQL statement: SELECT * FROM customer MyCustomer, myCustomer would be a table correlation.
Group	The GROUP BY clause
Having	The HAVING clause
Order	The ORDER BY clause
Fields	The fields of the SQL statement
Where	The WHERE clause

After a call to descontruct the SQL wil have an SQLStatement type of one of the following:

```
TSQLStatementType = (stUnknown, stSelect, stUpdate, stDelete, stInsert,
stAlter, stCreate);
```

These properties can also be edited and then Construct called so that the SQL is reassembled for use.

1.8.2.11.1 Properties

TAstaSQLParser

Correlations
DeleteTable
Fields
Group
Having
InsertTable
Order
SQL
SQLOptions
TableIsReadOnly
Tables

UpdateTable
Where

1.8.2.11.1.1 TAstaSQLParser.Correlations

TAstaSQLParser

Declaration
```
property Correlations: TStrings;
```

Description
Read Correlations to retrieve the list of aliases used in a SELECT statement.

1.8.2.11.1.2 TAstaSQLParser.DeleteTable

TAstaSQLParser

Declaration
```
property DeleteTable: string;
```

Description
Read DeleteTable to get the table name being deleted in a DELETE SQL statement.

1.8.2.11.1.3 TAstaSQLParser.Fields

TAstaSQLParser

Declaration
```
property Fields: TStrings;
```

Description
Read Fields to retrieve the list of result fields used in a SELECT SQL statement.

1.8.2.11.1.4 TAstaSQLParser.Group

TAstaSQLParser

Declaration
```
property Group: TStrings;
```

Description
Read Group to retrieve the list of GROUP BY fields used in a SELECT SQL statement.

1.8.2.11.1.5 TAstaSQLParser.Having

TAstaSQLParser

Declaration
```
property Having: string;
```

Description
Read Having to retrieve the HAVING clause of an SQL SELECT statement, if there is one.

1.8.2.11.1.6 TAstaSQLParser.InsertTable

TAstaSQLParser

Declaration
```
property InsertTable: string;
```

Description
Read InsertTable to get the table name being inserted into in an INSERT SQL statement.

1.8.2.11.1.7 TAstaSQLParser.Order

TAstaSQLParser

Declaration
```
property Order: TStrings;
```

Description
Read Order to retrieve the ORDER BY clause of an SQL SELECT statement, if there is one.

1.8.2.11.1.8 TAstaSQLParser.SQL

TAstaSQLParser

Declaration
```
property SQL: TStrings;
```

Description
Holds the SQL statement to be deconstructed, or created after a call to Construct.

1.8.2.11.1.9 TAstaSQLParser.SQLOptions

TAstaSQLParser

Declaration
```
property SQLOptions: TParseSQLOptions;
```

Description

1.8.2.11.1.10 TAstaSQLParser.TableIsReadOnly

TAstaSQLParser

Declaration
```
property TableIsReadOnly: Boolean;
```

Description
Returns True if the SQL statement reflects read only acitivty.

1.8.2.11.1.11 TAstaSQLParser.Tables

TAstaSQLParser

Declaration
```
property Tables: TStrings;
```

Description
Read Tables to retrieve the list of tables used in a SELECT SQL statement.

1.8.2.11.1.12 TAstaSQLParser.UpdateTable

TAstaSQLParser

Declaration
```
property UpdateTable: string;
```

Description
Read UpdateTable to get the table name being updated in an UPDATE SQL statement.

1.8.2.11.1.13 TAstaSQLParser.Where

TAstaSQLParser

Declaration
```
property Where: string;
```

Description
Read Where to retrieve the WHERE clause of an SQL statement.

1.8.2.11.2 Methods

TAstaSQLParser

Construct
Deconstruct
SQLStatetmentType

1.8.2.11.2.1 TAstaSQLParser.Construct

TAstaSQLParser

Declaration
```
procedure Construct;
```

Description
Creates an SQL statement from all of the deconstructed elements of a TAstaSQLParser.

1.8.2.11.2.2 TAstaSQLParser.Deconstruct

TAstaSQLParser

Declaration
```
procedure Deconstruct;
```

Description
Breaks the SQL statement down into the properties of the AstaSQLParser: Tables, Group, Having, Order, Fields, Where and UpdateTable.

1.8.2.11.2.3 TAstaSQLParser.SQLStatetmentType

TAstaSQLParser

Declaration
```
property SQLStatementType: TSQLStatementType;
```

Description
SQLStatementType returns one of the following:

Value	Meaning
stUnknown	The type of SQL statement is unknown.
stSelect	The SQL contains a SELECT statement.
stUpdate	The SQL contains an UPDATE statement.
stDelete	The SQL contains a DELETE statement.
stInsert	The SQL contains an INSERT statement.
stAlter	The SQL contains an ALTER statement.
stCreate	The SQL contains a CREATE statement.

1.8.3 Messaging

1.8.3.1 TAstaParamItem

Properties : Methods

Unit
AstaParamList

Description
A TAstaParamList is similar to the TParams property in the Delphi TQuery, except that each param can be streamed across the Internet and can contain any type of data including binary, other ParamLists and even datasets. Each Param member is of type TAstaParamItem which has the following properties:

```
Name: string;
ParamType: TAstaParamType;
DataType: TFieldType;
IsNull: Boolean;
Size: Integer;
```

1.8.3.1.1 Properties

TAstaParamItem

AsBlob
AsBoolean
AsCurrency
AsDataSet
AsDate
AsDateTime
AsDispatch
AsFloat
AsGUID
AsInteger
AsLargeInt
AsMemo
AsObject
AsParamList
AsSmallint
AsStream
AsString
AsTime
AsWord
DataType
IsInput
IsNull
IsOutput
Name
ParamType
Size
Text
Value

1.8.3.1.1.1 TAstaParamItem.AsBlob

TAstaParamItem

Declaration
```
property AsBlob: string;
```

Description
Set AsBlob to assign the value for a blob field to the parameter. Setting AsBlob will set the DataType property to ftBlob. Since long strings are just like buffers, binary values can safely be handled and are stored internally as strings.

1.8.3.1.1.2 TAstaParamItem.AsBoolean

TAstaParamItem

Declaration
```
property AsBoolean: Boolean;
```

Description
Set AsBoolean to assign the value for a boolean field to the parameter. Setting AsBoolean will set the DataType property to ftBoolean.

1.8.3.1.1.3 TAstaParamItem.AsCurrency

TAstaParamItem

Declaration
```
property AsCurrency: Currency;
```

Description

1.8.3.1.1.4 TAstaParamItem.AsDataSet

TAstaParamItem

Declaration
```
property AsDataSet: TDataSet;
```

Description
Set AsDataSet to assign the value for a dataset field (Delphi 4 and higher) to the parameter. Setting AsDataset will set the DataType property to ftDataSet in Delphi 4 and higher and to ftString in Delphi 3.

When using this as a function you are responsible for disposing of the dataset. Internally it calls the CloneDataSetToString and StringToDataSet.

1.8.3.1.1.5 TAstaParamItem.AsDate

TAstaParamItem

Declaration
```
property AsDate: TDateTime;
```

Description
Set AsDate to assign the value for a date field to the parameter. Setting AsDate will set the DataType property to ftDate;

1.8.3.1.1.6 TAstaParamItem.AsDateTime

TAstaParamItem

Declaration
```
property AsDateTime: TDateTime;
```

Description
Set AsDateTime to assign the value for a datetime field to the parameter. Setting AsDateTime will set the DataType property to ftDateTime.

1.8.3.1.1.7 TAstaParamItem.AsDispatch

TAstaParamItem

Declaration
```
property AsDispatch: Pointer;
```

Description

1.8.3.1.1.8 TAstaParamItem.AsFloat

TAstaParamItem

Declaration
```
property AsFloat: Double;
```

Description
Set AsFloat to assign the value for a float field to the parameter. Setting AsFloat will set the DataType property to ftFloat.

1.8.3.1.1.9 TAstaParamItem.AsGUID

TAstaParamItem

Declaration
```
property AsGUID: TGUID;
```

Description

1.8.3.1.1.10 TAstaParamItem.AsInteger

TAstaParamItem

Declaration
```
property AsInteger: Integer;
```

Description
Set AsInteger to assign the value for a integer field to the parameter. Setting AsInteger will set the DataType property to ftInteger.

1.8.3.1.1.11 TAstaParamItem.AsLargeInt

TAstaParamItem
This applies to Delphi 4 and BCB4 and higher only

Declaration
```
property AsLargeInt: LargeInt;
```

Description
Set AsLargeInt to assign the value for a large int field to the parameter. Setting AsLargeInt will set the DataType property to ftLargeInt.

1.8.3.1.1.12 TAstaParamItem.AsMemo

TAstaParamItem

Declaration
```
property AsMemo: string;
```

Description
Set AsMemo to assign the value for a memo field to the parameter. Setting AsMemo will set the DataType property to ftMemo

1.8.3.1.1.13 TAstaParamItem.AsObject

TAstaParamItem

Declaration
```
property AsObject: TObject;
```

Description
Set AsObject to assign the value for a pointer field to the parameter. Setting AsObject will set the DataType property to ftBlob.

Note: this is used internally by ASTA to pass some pointer values using ParamLists. These pointers are meaningless beyond the local machine boundry.

1.8.3.1.1.14 TAstaParamItem.AsParamList

TAstaParamItem

Declaration
```
property AsParamList: TAstaParamList;
```

Description

1.8.3.1.1.15 TAstaParamItem.AsSmallint

TAstaParamItem

Declaration
```
property AsSmallInt: SmalllInt;
```

Description
Set AsSmallInt to assign the value for a small int field to the parameter. Setting AsSmallInt will set the DataType property to ftSmallInt.

1.8.3.1.1.16 TAstaParamItem.AsStream

TAstaParamItem

Declaration
```
property AsStream: TMemoryStream;
```

Description
Set AsStream to assign the value for a memo field to the parameter. Setting AsStream will set the DataType property to ftBlob When using this property as a function, the caller is resposonsible for disposing of the stream.

Example

```
var
  m: TMemoryStream;
begin
  //The Param list creates a stream that *must* be disposed of.
  m := Params[0].AsStream;
  //do something
  m.Free;
```

```
end;
```

1.8.3.1.1.17 TAstaParamItem.AsString

TAstaParamItem

Declaration
```
property AsString: string;
```

Description
Set AsString to assign the value for a string field to the parameter. Setting AsString will set the DataType property to ftString.

1.8.3.1.1.18 TAstaParamItem.AsTime

TAstaParamItem

Declaration
```
property AsTime: TDateTime;
```

Description
Set AsTime to assign the value for a time field to the parameter. Setting AsTime will set the DataType property to ftTime.

1.8.3.1.1.19 TAstaParamItem.AsWord

TAstaParamItem

Declaration
```
property AsWord: Word;
```

Description
Set AsWord to assign the value for a word field to the parameter. Setting AsWord will set the DataType property to ftWord

1.8.3.1.1.20 TAstaParamItem.DataType

TAstaParamItem

Declaration
```
property DataType: TFieldType;
```

Description
Indicates the type of field whose value the parameter represents. DataType is set automatically when a value is assigned to the parameter.

Read DataType to discover the type of data that was assigned to the parameter. Each possible value of DataType corresponds to a type of database field.

1.8.3.1.1.21 TAstaParamItem.IsInput

TAstaParamItem

Declaration
```
function TAstaParamItem.IsInput: Boolean;
```

Description

A quick way to check if a TAstaParamItem is an input parameter.

1.8.3.1.1.22 TAstaParamItem.IsNull

TAstaParamItem

Declaration
```
property IsNull: Boolean;
```

Description
Indicates whether the value assigned to the parameter is NULL (blank). Inspect IsNull to discover if the value of the parameter is NULL, indicating the value of a blank field. NULL values can arise in the following ways:

1. Assigning the value of another, NULL, parameter.
2. Assigning the value of a blank field.
3. Calling the Clear method.

1.8.3.1.1.23 TAstaParamItem.IsOutput

TAstaParamItem

Declaration
```
function TAstaParamItem.IsOutput: Boolean;
```

Description
A quick way to check if a TAstaParamItem is an output parameter.

1.8.3.1.1.24 TAstaParamItem.Name

TAstaParamItem

Declaration
```
property Name: string;
```

Description
Use Name to identify a particular parameter within a TParams object. Name is the name of the parameter, not the name of the field the value of which the parameter represents.

1.8.3.1.1.25 TAstaParamItem.ParamType

TAstaParamItem

Declaration
```
property ParamType: TAstaParamType
```

Description
Objects that use TParam objects to represent field parameters set ParamType to indicate the use for the parameter. ParamType must be one of the following values:

Value	Meaning
ptUnknown	Unknown or undetermined. Before executing a stored procedure, the application must set parameters of this type to another kind.
ptInput	Used to input a field value. Identifies a parameter used to pass values to a stored procedure for processing.
ptOutput	Used to output a field value. Identifies a parameter used by a stored procedure to return values to an application.
ptInputOutput	Used for both input and output.
ptResult	Used as a return value. Identifies a parameter used by a stored procedure to return an error or status value. A stored procedure can only have one parameter of type ptResult.

1.8.3.1.1.26 TAstaParamItem.Size

TAstaParamItem

Declaration
```
Property Size: Integer;
```

Description
Size of the param value. For blobs, memos and streams this will return the actual stored size.

1.8.3.1.1.27 TAstaParamItem.Text

TAstaParamItem

Declaration
```
property Text: string;
```

Description
Set the Text property to assign the value of the parameter to a string without changing the DataType. Unlike the AsString property, which sets the value to a string and changes the DataType, setting the text property converts the string to the datatype of the parameter, and sets the value accordingly. Thus, use AsString to treat the parameter as representing the value of a string field. Use Text instead when assigning a value that is in string form, when making no assumptions about the field type. For example, Text is

useful for assigning user data that was input using an edit control.

Reading the Text property is the same as reading the AsString property.

1.8.3.1.1.28 TAstaParamItem.Value

TAstaParamItem

Declaration
```
property Value: Variant;
```

Description
Represents the value of the parameter as a Variant. Use Value in generic code that manipulates the values of parameters without needing to know the field type the parameters represent.

1.8.3.1.2 Methods

TAstaParamItem

Add
AssignField
AssignParam
AssignStdParam
AssignTo
AssignToStdParam
Clear
FindParam
GetDataSize
IsInput
LoadFromFile
LoadFromStream
ParamByName
SetBlobData

1.8.3.1.2.1 TAstaParamItem.Add

TAstaParamItem

Declaration
```
function Add: TAstaParamItem;
```

Description
Adds a TAstaParamItem and returns it as a result. See FastAdd for a fast, easy way to add parameter items to a TAstaParamList.

Example

```
with ParamList.Add do begin
  Name := 'MyParam';
  AsString := 'Hellow World';
  ParamType := ptInput;
end;
```

1.8.3.1.2.2 TAstaParamItem.AssignField

TAstaParamItem

Declaration
```
procedure AssignField(Field: TField);
```

Description
Assigns the value of the TAstaParamItem from a TField.

1.8.3.1.2.3 TAstaParamItem.AssignParam

TAstaParamItem

Declaration
```
procedure AssignParam(Dest: TAstaParamItem);
```

Description
Assigns all the properties of the TAstaParamItem to Dest (TAstaParamItem).

1.8.3.1.2.4 TAstaParamItem.AssignStdParam

TAstaParamItem

Declaration
```
procedure AssignStdParam(Src: TParam);
```

Description
Allows you to assign a standard Delphi TParam to a TAstaParamItem.

1.8.3.1.2.5 TAstaParamItem.AssignTo

TAstaParamItem

Declaration
```
procedure TAstaParamItem.AssignTo(Dest: TPersistent);
```

Description
If Dest is a TField, AssignTo assigns the Value of the TAstaParamItem to it. Otherwise it expects Dest to be a TAstaParamItem.

1.8.3.1.2.6 TAstaParamItem.AssignToStdParam

TAstaParamItem

Declaration
```
procedure AssignToStdParam(Dest: TParam);
```

Description
Allows you to assign a TAstaParamItem to a standard Delphi TParam.

1.8.3.1.2.7 TAstaParamItem.Clear

TAstaParamItem

Declaration
```
procedure Clear;
```

Description
Empties the AstaParamList.

1.8.3.1.2.8 TAstaParamItem.FindParam

TAstaParamItem

Declaration
```
function FindParam(ParamName: string): TAstaParamItem;
```

Description
This is similar to the TParam.FindParam and returns nil if it does not find the ParamName.

1.8.3.1.2.9 TAstaParamItem.GetDataSize

TAstaParamItem

Declaration
```
function GetDataSize: Integer;
```

Description
Returns the size in bytes of the internal storage used by AstaParamItems.

1.8.3.1.2.10 TAstaParamItem.IsBlob

TAstaParamItem

Declaration
```
function IsBlob: Boolean;
```

Description

1.8.3.1.2.11 TAstaParamItem.LoadFromFile

TAstaParamItem

Declaration
```
procedure TAstaParamItem.LoadFromFile(FileName: string);
```

Description
Assigns a TAstaParamItem value from data contained in a file. Internally this creates a TFileStream, does a load from file and stores the value using TAstaParamItem.AsStream.

1.8.3.1.2.12 TAstaParamItem.LoadFromStream

TAstaParamItem

Declaration

```
procedure TAstaParamItem.LoadFromStream(Stream: TStream);
```

Description
Assigns the data from a stream to the TAstaParamItem value calling AsStream which sets the DataType to ftBlob.

1.8.3.1.2.13 TAstaParamItem.ParamByName

TAstaParamItem

Declaration
```
function ParamByName(ParamName: string): TAstaParamItem;
```

Description
This is similar to the FindParam function and returns NIL if the ParamName is not found.

1.8.3.1.2.14 TAstaParamItem.SetBlobData

TAstaParamItem

Declaration
```
procedure TAstaParamItem.SetBlobData(Buffer: Pointer; Size: Integer);
```

Description
Sets the datatype to ftBlob and moves the data pointed to by Buffer to the internal storage of the TAstaParamItem.

1.8.3.2 TAstaParamList

Properties : Methods

Unit
AstaParamList

Description
A TAstaParamList is similar to the TParams property in the Delphi TQuery except that they can be streamed across the internet and can contain any type of data including binary, other ParamLists and even datasets. Each Param member is of type TAstaParamItem which has the following properties:

```
Name: string;
ParamType: TAstaParamType;
DataType: TFieldType;
Null: Boolean;
Size: Integer;
```

This allows the TAstaParamList to be used just like the Delphi TParams. Here are some examples:

```
ParambyName('CustomerNumber').AsInteger := 101;
Params[0].AsInteger := 101;
```

Use the TastaParmList.FastAdd to add any kind of data (excluding blob data). It takes a variant as an argument and calls TAstaParamList.Add to automatically add and set the data according to the variant datatype.

1.8.3.2.1 Properties

TAstaParamList

Items

1.8.3.2.1.1 TAstaParamList.Items

TAstaParamList

Declaration
```
property Items[Index: Integer]: TAstaParamItem;
```

Description
Items is the default property of the TAstaParamList so you can use it like ParamList[i] to access an individual TAstaParamItem.

1.8.3.2.2 Methods

TAstaParamList

AddNamedParam
AddOneParamItem
AsDisplayText
AssignValues
AsTokenizedString
ConstantAdd
CopyList
CopyParams
CopyParamsType
CreateFromString
CreateParam
DeleteParam
FastAdd
FindParam
LoadFromTokenizedString
ParamByName
SaveToFile
SaveToStream
UpdateAddParams

1.8.3.2.2.1 TAstaParamList.AddNamedParam

TAstaParamList

Declaration
```
procedure AddNamedParam(AName: string; Value: Variant);
```

Description
This procedure allows you to easily values to an AstaParamList and to supply a name for the param. In versions of Delphi beyond Delphi 3 the FastAdd method is overloaded so you can also use that.

1.8.3.2.2.2 TAstaParamList.AddOneParamItem

TAstaParamList

Declaration
```
procedure AddOneParamItem(Value: TAstaParamItem);
```

Description
Adds one TAstaParamItem to the list. Used in Delphi 3. with Delphi 5 and up use the overloaded FastAdd(Value: TAstaParamItem).

1.8.3.2.2.3 TAstaParamList.AsDisplayText

TAstaParamList

Declaration
```
function AsDisplayText: string;
```

Description
Used in Server logs to be able to display the ParamNames and Values as Strings. For blobs/memos the size of the blob is displayed.

1.8.3.2.2.4 TAstaParamList.AssignValues

TAstaParamList

Declaration
```
procedure AssignValues(Dest: TAstaParamList);
```

Description
If a Param named Dest.Param exists in the AstaParamList the Dest will be assigned to that Param. If the Param does not exist it will be ignored.

1.8.3.2.2.5 TAstaParamList.AsTokenizedString

TAstaParamList

Declaration
```
function AsTokenizedString(InputsOnly: Boolean): string;
```

Description
This saves a TAstaParamList as a string so that it can be easily recreated later using CreateFromString.

1.8.3.2.2.6 TAstaParamList.ConstantAdd

TAstaParamList

Declaration
```
procedure ConstantAdd(const Values: array of const);
```

Description
Allows an Array of Const to be added to the ParamList.

1.8.3.2.2.7 TAstaParamList.CopyList

TAstaParamList

Declaration
```
function CopyList: TAstaParamList;
```

Description
This makes a copy of the TAstaParamList.

1.8.3.2.2.8 TAstaParamList.CopyParams

TAstaParamList

Declaration
```
procedure CopyParams(Dest: TAstaParamList; ClearIt: Boolean);
```

Description
This will copy all of the AstaParamItemsto Dest and optionally clear the destination ParamList.

1.8.3.2.2.9 TAstaParamList.CopyParamsType

TAstaParamList

Declaration
```
procedure CopyParamsType(Dest: TAstaParamList; Clearit: Boolean; Accepted:
  TAstaParamTypes);
```

Description
Makes a copy of the AstaParamList. If Accepted is [] all param types will be included in the copy. Optionally only certain types (example [ptInput]) can be included for the copy.

1.8.3.2.2.10 TAstaParamList.CreateFromString

TAstaParamList
This replaces the previous call of CreateFromTokenizedString.

Declaration
```
constructor CreateFromString(S: string; Token: TAstaParamListCreateToken);
TAstaParamListCreateToken = (tcToken, tcServer, tcJava, tcPalm);
```

Description
This is used quite frequently by ASTA servers internally to receive incoming strings from ASTA clients and to convert the strings to TAstaParamLists. Since ASTA ParamLists can contain any kinds of data, using the CreateFromString method along with the AsTokenizedString method allows you to even have TAstaParamLists where each item is another TAstaParamList or to store AstaParamLists as strings in blob fields in a DataSet.

To save a TAstaParamList as a string see AsTokenizedString. Most often you would use this method with the tcToken param.

Example

```
var
  p: TAstaParamList;
```

```
begin
  p := TAstaParamList.CreateFromString(SomeStringSavedAsTokenizedString,
  tcToken);
end;
```

1.8.3.2.2.11 TAstaParamList.CreateParam

TAstaParamList

Declaration

```
function CreateParam(FldType: TFieldType; const ParamName: string;
  AParamType: TAstaParamType): TAstaParamItem;
```

Description

This method creates a param and adds it to the AstaParamList. For more simple calls check out the overloaded method FastAdd.

1.8.3.2.2.12 TAstaParamList.DeleteParam

TAstaParamList

Declaration

```
function DeleteParam(ParamName: string): Boolean;
```

Description

Deletes and frees a ParamItem if the name of the param exists. Returns True if the Param exists.

1.8.3.2.2.13 TAstaParamList.FastAdd

TAstaParamList

Declaration

```
procedure FastAdd(Value: Variant); overload;
procedure FastAdd(AName: string; Value: Variant); overload;
procedure FastAdd(AName: string; ADataType: TFieldType; AValue: Variant);
  overload;
procedure FastAdd(const AName: string; ADataType: TFieldType; AValue:
  Variant; AParamType: TAstaParamType); overload;
procedure FastAdd(Value: TAstaParamItem); overload;
```

Description

This procedure allows you to easily values to an AstaParamList. Internally the Add method is called, a name for the parameter is assigned and the Value property is used to set the incoming Value.

Example

```
FastAdd('Hello World'); //adds string
FastAdd(Now); //adds datetime
FastAdd(4.33); //adds float
FastAdd(5);  //adds smallint
FastAdd(True); //adds Boolean
```

Streams cannot be added this way, but you can use FastAdd('') and then Params[Count - 1].AsStream to add a stream.

Variations
```
procedure FastAdd(Value: TAstaParamItem);
```
This allows you to add a parameter whose value is taken from another TAstaParamItem.

```
procedure FastAdd(AName: string; Value: Variant);
```
This allows you to add a named parameter and value in the same step for Delphi 4 and 5. For Delphi 3, use the procedure AddNamedParam(AName: string; Value: Variant).

```
procedure FastAdd(AName: string; ADataType: TFieldtype; AValue: Variant);
```
This allows you to add a named parameter, datatype and value in the same step for Delphi 4 and 5.

1.8.3.2.2.14 TAstaParamList.FindParam

TAstaParamList

Declaration
```
function FindParam(const ParamName: string): TAstaParamItem;
```

Description
Just like the TParam.FindParam function. Returns a pointer to it if the param is found, otherwise returns nil.

1.8.3.2.2.15 TAstaParamList.LoadFromTokenizedString

TAstaParamList

Declaration
```
procedure LoadFromTokenizedString(const S: string);
```

Description
ParamLists can be saved as Strings. This allows for ParamLists to contain lists of full ParamLists. LoadFromTokenizedString can load a Paramlist from a string that has been saved using AsTokenizedString.

To create an AstaParamList from a string use TAstaParamList.CreateFromString(const S: string; Token: TAstaParamListCreateToken).

1.8.3.2.2.16 TAstaParamList.ParamByName

TAstaParamList

Declaration
```
function ParamByName(const ParamName: string): TAstaParamItem;
```

Description
Like the standard TParams.ParamByName function, returns a ParamItem by it's name or raises an exception if not found.

1.8.3.2.2.17 TAstaParamList.SaveToFile

TAstaParamList

Declaration

```
procedure SaveToFile(FileName: string);
```

Description
Save a ParamList to a file.

1.8.3.2.2.18 TAstaParamList.SaveToStream

TAstaParamList

Declaration
```
procedure SaveToStream(Stream: TStream);
```

Description
Save a ParamList to a stream.

1.8.3.2.2.19 TAstaParamList.SaveToString

TAstaParamList

Declaration
```
function SaveToString: string;
```

Description
Saves a ParamList to a String. The same as calling AsTokenizedString(False).

1.8.3.2.2.20 TAstaParamList.UpdateAddParams

TAstaParamList

Declaration
```
procedure UpdateAddParams(Dest: TAstaParamList);
```

Description
This will add to Dest any params that don't exist or update any params that exist by searching for the param using FindParam.

1.8.3.3 TAstaMQManager

1.8.4 Additional Types

1.8.4.1 TAstaIndexes

Properties : Methods : Events

Unit
AstaIndexes

Declaration
```
type TAstaIndexes = class(TCollection)
```

Description
Asta datasets can have indexes defined so that multiple sort orders can be defined. Indexes can be added at design time using the Indexes property or at runtime by calling AddIndex or AddIndexFields.

1.8.4.1.1 Properties

TAstaIndexes

Active
BMKActive
DataSet
Items
Ordered
SelectedIndex
SelectedIndexFields
SelectedIndexName

1.8.4.1.1.1 TAstaIndexes.Active

TAstaIndexes

Declaration
```
property Active: Boolean;
```

Description

1.8.4.1.1.2 TAstaIndexes.BMKActive

TAstaIndexes

Declaration
```
property BMKActive: Boolean;
```

Description

1.8.4.1.1.3 TAstaIndexes.DataSet

TAstaIndexes

Declaration
```
property DataSet: TDataSet;
```

Description

1.8.4.1.1.4 TAstaIndexes.Items

TAstaIndexes

Declaration
```
property Items[AIndex: Integer]: TAstaIndex;
```

Description

1.8.4.1.1.5 TAstaIndexes.Ordered

TAstaIndexes

Declaration
```
property Ordered: Boolean;
```

Description

1.8.4.1.1.6 TAstaIndexes.SelectedIndex

TAstaIndexes

Declaration
```
property SelectedIndex: TAstaIndex;
```

Description

1.8.4.1.1.7 TAstaIndexes.SelectedIndexFields

TAstaIndexes

Declaration
```
property SelectedIndexFields: string;
```

Description

1.8.4.1.1.8 TAstaIndexes.SelectedIndexName

TAstaIndexes

Declaration
```
property SelectedIndexName: string;
```

Description

1.8.4.1.2 Methods

TAstaIndexes

LoadFromStream
SaveToStream

1.8.4.1.2.1 TAstaIndexes.LoadFromStream

TAstaIndexes

Declaration
```
procedure LoadFromStream(AStream: TStream);
```

Description

1.8.4.1.2.2 TAstaIndexes.SaveToStream

TAstaIndexes

Declaration
```
procedure SaveToStream(AStream: TStream);
```

Description

1.8.4.1.3 Events

TAstaIndexes

OnChanged
OnChanging

1.8.4.1.3.1 TAstaIndexes.OnChanged

TAstaIndexes

Declaration
```
property OnChanged: TAstaIndexChangeEvent;
```

Description

1.8.4.1.3.2 TAstaIndexes.OnChanging

TAstaIndexes

Declaration
```
property OnChanging: TAstaIndexChangeEvent;
```

Description

1.8.4.2 TAstaMetaData

Unit
AstaDBTypes

Declaration
```
TAstaMetaData = (mdNormalQuery, mdTables, mdIndexes, mdFields, mdViews,
  mdStoredProcs, mdForeignKeys, mdSystemTables, mdDBMSName, mdPrimeKeys,
  mdStoredProcColumns, mdServerDataSets, mdCustomDataSets,
  mdBusinessObjects, mdProviders, mdBusObjectParams, mdProviderParams,
  mdVCLFields, mdOtherMetaData, mdBusinessObjectNames, mdUserList,
  mdAliasedBizObjectMethods);
```

Description

Value	Meaning
mdNormalQuery	
mdTables	
mdIndexes	
mdFields	
mdViews	
mdStoredProcs	
mdForeignKeys	
mdSystemTables	
mdDBMSName	
mdPrimeKeys	
mdStoredProcColumns	
mdServerDataSets	
mdCustomDataSets	
mdBusinessObjects	
mdProviders	
mdBusObjectParams	
mdProviderParams	
mdVCLFields	
mdOtherMetaData	
mdBusinessObjectNames	
mdUserList	
mdAliasedBizObjectMethods	

1.8.4.3 TAstaParamType

Unit
AstaParamList

Declaration
```
type TAstaParamType = (ptUnknown, ptInput, ptOutput, ptInputOutput,
```

```
ptResult);
```

Description
These correspond to the same ordinals as the TParams as defined in db.pas in Delphi 4 and up.

1.8.4.4 TAstaProtocol

Unit
AstaStringLine

Applies to
TAstaServerSocket
TAstaClientSocket

Declaration
```
TAstaProtocol = (apTCPIP, apHTTPTunnel, apISAPI)
```

Description
ASTA servers and clients use a proprietary messaging format with the default transport using TCP/IP. To traverse firewalls ASTA TCP/IP messaging can be made to appear as HTTP messages or actually become HTTP messsages. There is no need to set this property on the AstaServerSocket as ASTA servers can serve TCP/IP and HTTP clients unchanged. The only requirement is that to support stateless HTTP clients, servers must be run using Pooled or Smart Threading.

ASTAClientSockets can call serveral helper methods and run time dialogs to configure ASTA clients to use protocols other than TCP/IP.

SetForIsapiUse
SetForProxyUse
SetupForSocksServer
Firewalls

1.8.4.5 TAstaSelectSQLOptionSet

Unit
AstaDBTypes

Declaration
```
type TAstaSelectSQLOptionSet = set of TAstaSelectSQLOptions;

type TAstaSelectSQLOptions = (soFetchMemos, soFetchBlobs, soPackets,
soFieldProperties, soCyclopsSQL, soCompress, soIgnoreAutoIncrement,
soExecStoredProc, soFetchBytes, SoDelayed);
```

Description

soFetchMemos	This will fetch all the ftMemo fields in selects and also allow them to be part of any Insert or Edit's applied to the server. Note that since all fields in the select statement that translate to ftmemos will be fetched, using this feature could cause large memos to be streamed from the server to the client.
soFetchBlobs	This will fetch all ftBlob or ftGraphic fields similar to the way that soFetchMemos works will the same performance considerations.
soPackets	When this is set, and the RowsToReturn is a positive number, and the AstaServer is running in the PersistentSessions threading model, this will cause the original select to be opened as a cursor on the server bringing back only RowsToReturn rows initially. See the discussion on GetNextPackets.
soCompress	When this is set the select statement will be compressed on the server using AstaCompression and and will be uncompressed on the client. This is ignored if you are have compression set on in the TAstaClientSocket. You may gain significant performance benefits by using this property on larger queries and if your server has sufficient CPU.
soIgnoreAutoIncrement	Used to allow AutoIncrement fields to be updated.
soExecStoredProc	Some stored procedures return result sets and others do not. If a stored Procedure does not return a result set then call ExecSQL which is the same as setting soExecStoredProc to true. This will pass the NoResultSet:Boolean to the AstaServerSocket.OnStoredProcedure event as true.
soCyclopsSQL	This allows for select statements to be written, one for each item in the SQL:TStrings property and a result set will be returned that contains the SQL and an AstaDataSet saved as a string in a blob field. Used internally by ExpresswaySQL.
soFetchBytes	This will fetch all ftBytes or ftVarBytes fields similar to the way that soFetchMemos works will the same performance considerations.
soDelayed	This works only with client side SQL and when the AstaServer is running in Pooled Sessions threading and allows for...

1.8.4.6 TAstaServerSQLOptions Type

Unit
AstaSQLUtils

Declaration
```
type TAstaServerSQLOptionTypes = (ssUseQuotesInFieldNames,
  ssTableNameInInsertStatments, ssBooleanAsInteger,
  ssUSDatesForLocalSQL, ssTerminateWithSemiColon,
```

```
ssNoQuotesInDates, ssDoubleQuoteStrings, ssUseISInNullCompares);
type TAstaServerSQLOptions = set of TAstaServerSQLOptionTypes;
```

Description

Value	Meaning
ssUseQuotesInFieldNames	Used for Paradox or any database that allows spaces in field names.
ssTableNameInInsertStatments	Databases like Access need this set to false.
ssBooleanAsInteger	MS SQL Server has bit fields that map to ftBoolean in the VCL but then SQL must be generated as Integers (eg. 0 or 1).
ssUSDatesForLocalSQL	Interbase, Paradox and xBase using the BDE don't like international dates so this forces the server to receive US formatted dates.
ssTerminateWithSemiColon	Access likes it's SQL statements to be terminated with semi-colons.
ssNoQuotesInDates	Use this if you don't want to use single quotes in dates and datetimes.
ssDoubleQuoteStrings	Use this if your strings don't need single quotes but double quotes.
ssUseISInNullCompares	Paradox doesn't like = null and wants IS null.

1.8.4.7 TAstaUpdateMethod

Unit
AstaDBTypes

Declaration
```
type TAstaUpdateMethod = (umManual, umCached, umAfterPost);
```

Description
Determines when SQL is posted to the server. umAfterPost will fire after a record is posted. umCached requires the ApplyUpdates method to be called.

1.8.4.8 TAstaUtilityEvent

Unit
AstaServer

Declaration
```
type TAstaUtilityEvent = procedure(Sender: TObject;ServerSocket:
  TAstaServerSocket; ClientSocket: TCustomWinSocket; Params:
```

```
  TAstaParamList) of object;
```

Description
The AstaServerSocket is non-blocking and event driven and provides excellent performance without using threads. ASTA database activity can be threaded and to scale an ASTA server, the Pooled Threading Model is recommended. See the book *COM + and the Battle for the Middle Tier* by Roger Sessions (Wiley Press) for a description of scalable servers that applies quite accurately to ASTA servers running the Pooled Sessions threading model.

One of the design goals of ASTA was to provide scalable threaded servers without having ASTA developers to have to create any threads themselves. Since ASTA 2 there have been threaded messaging calls that allow for ASTA messaging to be used in threads on the server. Immediately ASTA users started to push the envelope on asynchronous database access. The problem arose that one remote client would want to be able to have multiple database queries executing on a server, and if left un-checked, this could result in one client using many, many database sessions.

In Asta 2.108 an attempt was made to cache asynchronous threaded database calls so that a limit could be set on the amount of concurrent database sessions used by any remote user. To send a message to an ASTA client, normally a ClientSocket (TCustomWinSocket) is used. Internally, ASTA uses the ClientSocket to lookup the UserRecord from the UserList of the TAstaServerSocket. In order to make this thread safe, instead of using the UserList, ASTAThreads now make a full copy of the UserRecord that can be used within threaded util calls in order to make it thread safe. In addition, critical sections have been added to code where it was not needed before when using non-blocking event driven sockets.

The prototypes of the ThreadedUtil events have changed. In the past, the Sender (TObject) parameter passed was the AstaServerSocket, and now it will be a TAstaThread. This will allow you to use the UserRecord of the AstaThread to send messages.

The following shows the use of a ThreadedDBUtility event from an ASTA server. Notice the ServerSocket.SendCodedParamListSocket call which uses the ThreadedClient copy of the UserRecord. Note: If a client is disconnected at this point the thread will be marked as terminated, so perhaps a call to **if not** TAstaThread(Sender).HasTerminted **then** Exit is needed before the actual send is made. The one scenario where we still seem to have problems is when there are queued UtilityEvents that have not been launched, and the client disconnects.

```
procedure TForm1.TestUtilSQLEvent(Sender: TObject;
  ServerSocket: TAstaServerSocket; ClientSocket: TCustomWinSocket;
  Query: TComponent; Params: TAstaParamList);
var
  p: TAstaParamList;
  s: string;
begin
  //the Sender is now a TAstaThread
  try
    TQuery(Query).Close;
    TQuery(Query).SQL.Text := Params[0].AsString;
    TQuery(Query).Open;
  except
    ServerSocket.SendExceptionToClient(ClientSocket,
      Exception(exceptObject).Message);
```

```
    Exit;
      end;
      p := TAstaParamList.Create;
      p.FastAdd(IntToStr(TQuery(Query).RecordCount) + ' records
    retrieved.');
      p.FastAdd(Tquery(Query).SQL.Text);
      p.FastAdd(CloneDataSetToString(Query as TQuery));
      //don't call the AstaServerSocket1 as we are in a thread
      //use the passed in ServerSocket
      ServerSocket.SendCodedParamListSocket(TAstaThread(Sender).
        ThreadedClient, 1550, P);
      //Asta disposes of the query
      //if you want additional components you can call
      //ThreadedDBSupplyQuery since you now have the Sender TAstaThread
      //and the client socket
      p.Free;
    end;
```

1.8.4.9 TAstaWebServer

Unit
AstaClientSocket

Declaration
```
type TAstaWebServer = class(TPersistent)
```

Description
ASTA supports HTTP clients using the AstaClientSocket. The AstaClientSocket.WebServer property is a property that allows the configuration and control of HTTP settings. The AstaClientSocket can optionally use the Microsoft WinInet.dll that comes with Internet Explorer. WinInet uses any registry settings defined for Internet Explorer which include any proxy and Socks settings. Setting the WebServer.WinInet property (Boolean) to true is the recommended way to traverse firewalls with ASTA. If Internet explorer can browser the Internet, your ASTA WinInet enabled client will be able to communicate with your remote ASTA server through a web server and Astahttp.dll

Normal TCP/IP ASTA clients can support "server push", authentication, and ASTA messaging. When running stateless HTTP the landscape changes a bit. ASTA does provide features to allow stateless HTTP clients to use features that ASTA users have been accustomed to.

Authentication: Stateless Cookies
Normal ASTA TCP/IP client applications support a login process that allows for remote users to be authenticated against ASTA servers with a username and password, and to also send extra params to the server and receive params back from the server on the OnLoginParamsEvent. In addition there is a UserList that maintains Username, Application Name, Connect Time and other information on the AstaServerSocket. When running stateles of course there is no UserList. ASTA supports stateless UserLists with a binarytree available as the AstaServerSocket.StatelessUserList or with the AstaMessageQueue component.

The ASTA Cookies tutorial has an example server and client to show how authentication can be done with ASTA and stateless clients.

Asta stateless support via HTTP requires a web server to run like Microsoft IIS (internet Information Server). Another good choice for testing is the Omini Web Server available from www.omnicron.ca.

1. Start the WebServer.
2. Copy AstaHttp.dll into a directory that has the rights to execute an ISAPI dll. For IIS the default is the scripts directory.
3. Compile and Run the AstaHttpTestServer that comes with the ASTA Cookies tutorial.
4. Compile and run the AstaHttpTestClient.dpr that comes in in the Asta Cookies Tutorial.

Server Side
The AstaServerSocket has an AddCookie property which is of type TAstaServerCookieOption.

TAstaServerCookieOption = (coNoCookie, coStatelessUserList, coUserCoded);

In order to tunnel via HTTP, ASTA binary messages are wrapped using the HTTP format so that they can be disguised as normal browser traffic in order to traverse any client firewall. When cookies are used, additional authentication information is included in the ASTA messaging format.

Value	Meaning
coNoCookie	No cookie information
coStatelessUserList	A binary tree is used to authenticate clients and maintain cookie information. The AstaServerSocket has a property of StatelessUserList of type TAstaStatelessUserList.
coUserCoded	The user is responsible for implementing their own cookie mechanism. in the Cookies tutorial the AstaMessageQueue Manager is used for this.

In the tutorial, the AddCookie property is set to coStatelessUserlist. Since HTTP is a stateless protocol, there is no real user list for normal ASTA TCP/IP clients. When users connect, they will provide username and password information so that the server can authenticate them (this happens in the AstaServerSocket.OnClientAuthenticate event). A "Cookie" or Cardinal value will be returned to the client that can be used in future accesss of the server. This allows for a fast/efficient way to authenticate remote clients on each connect to the server.

1.8.4.9.1 Properties

TAstaWebServer

AddCookie
AstaHttpDll
AstaServerAddress
AstaServerPort
Authenticate
ProxyPassword
ProxyUserName
SSLOptions
Timeout
UseSSL
UseWebServer
WebServerAddress
WebServerPort
WinInet
WinInetToStatusBar

1.8.4.9.1.1 TAstaWebServer.AddCookie

TAstaWebServer

Declaration
```
property AddCookie: Boolean;
```

Description
Adds a cookie to client packets.

1.8.4.9.1.2 TAstaWebServer.AstaHttpDll

TAstaWebServer

Declaration
```
property AstaHttpDll: string;
```

Description
The location and name of Astahttp.dll, example "scripts/Astahttp.dll".

1.8.4.9.1.3 TAstaWebServer.AstaServerAddress

TAstaWebServer

Declaration
```
property AstaServerAddress: string;
```

Description
The address of the ASTA server relative of the web server. This probably will not be an internet address but an address on the local network from the web server.

1.8.4.9.1.4 TAstaWebServer.AstaServerPort

TAstaWebServer

Declaration
```
property AstaServerPort: Word;
```

Description
The port of the ASTA Server.

1.8.4.9.1.5 TAstaWebServer.Authenticate

TAstaWebServer

Declaration
```
property Authenticate: Boolean;
```

Description
True if your web server requires authentication.

Note: When using WinInet this property probably does not need to be set as WinInet uses the settings from IE (Internet Explorer).

1.8.4.9.1.6 TAstaWebServer.ProxyPassword

TAstaWebServer

Declaration
```
property ProxyPassword: string;
```

Description
The proxy password.

Note: When using WinInet this property probably does not need to be set as WinInet uses the settings from IE (Internet Explorer).

1.8.4.9.1.7 TAstaWebServer.ProxyUserName

TAstaWebServer

Declaration
```
property ProxyUserName: string;
```

Description
The proxy username.

Note: When using WinInet this property probably does not need to be set as WinInet uses the settings from IE (Internet Explorer).

1.8.4.9.1.8 TAstaWebServer.SSLOptions

TAstaWebServer

Declaration
```
property SSLOptions: TSSLOptions;
```

Description
Allows for WinINET SSL options to be configured.

1.8.4.9.1.9 TAstaWebServer.Timeout

TAstaWebServer

Declaration
```
property Timeout: Word;
```

Description
Time in seconds to block or wait for the server to return a reaponse. Set to a higher value if you are sending lots of data over the wire.

Note: the ASTA server "knows" that a client is using HTTP and will disconnect the client as soon as any server side process is completed. So setting a higher value will not affect performance.

1.8.4.9.1.10 TAstaWebServer.UseSSL

TAstaWebServer

Declaration
```
property UseSSL: Boolean;
```

Description
Uses SSL (requires a certificate on the server).

1.8.4.9.1.11 TAstaWebServer.UseWebServer

TAstaWebServer

Declaration
```
property UseWebServer: Boolean;
```

Description
Determines whether settings to HTTP tunnel should be activated.

1.8.4.9.1.12 TAstaWebServer.WebServerAddress

TAstaWebServer

Declaration
```
property WebServerAddress: string;
```

Description
The IP address or host of the web server (eg IIS).

1.8.4.9.1.13 TAstaWebServer.WebServerPort

TAstaWebServer

Declaration
```
property WebServerPort: Word;
```

Description
Web Server port. Traditionally port 80.

1.8.4.9.1.14 TAstaWebServer.WinInet

TAstaWebServer

Declaration
```
property WinInet: Boolean;
```

Description
Setting WinInet to True tells TAstaWebServer to use WinInet.

1.8.4.9.1.15 TAstaWebServer.WinInetToStatusBar

TAstaWebServer

Declaration
```
property WinInetToStatusBar: Boolean;
```

Description
Writes WinInet status messages to a TAstaStatusBar.

1.8.4.10 TRefetchStatus

Unit
AstaDBTypes

Declaration
```
type TRefetchStatus = (rfNone, rfAutoIncPrimeKey, ftNeedsPrimeFields);
```

Description

1.8.4.11 TThreadDBAction

Unit
AstaDBTypes

Declaration
```
type TThreadDbAction = (ttSelect, ttExec, ttStoredProc, ttMetaData,
ttBlobFetch, ttTransaction, TransactionStart,
ttCustom, ttDataModule);
```

Description
These values are passed internally by ASTA to the ThreadedDBSupplyQuery event so that ASTA servers can have components necessary to complete database tasks and is also used in the ThreadedDBUtilityEvent allow you to request any kind of database component handled by your ThreadedDBSupplyQuery to be used in a thread in ASTA messaging calls.

1.8.4.12 TUpdateSQLMethod

Unit
AstaClientDataSet

Declaration
```
type TUpdateSQLMethod = (usmNoTransactions, usmUseSQL,
usmServerTransaction, usmCommitAnySuccess);
```

Description
For use in the TAstaClientDataSet.ApplyUpdates(TransactionMethod: TUpdateSQLMethod).

usmNoTransactions	Do not use a transaction.
usmUseSQL	In ODBC and other databases, transactions can be started using an SQL command. With ODBC it is 'Begin Transaction' and to end the transaction you use 'COMMIT'. See the TAstaClientSocket.SQLTransactionStart and TAstaClientSocket.SQLTransactionEnd properties if you need to
usmServerTransaction	This is the preferred method of using ApplyUpdates and calls whatever component method that the database in use on the ASTA server uses to start a transaction. In the BDE it uses the TDatabase.StartTransaction. ASTA servers implement this by writing a method for the TAstaServerSocket.OnTransactionBegin event. See the TAstaServerSocket.OnTransactionEnd event which is called on the ASTA server after a transaction has been
usmCommitAnySuccess	Commit with Any Success means that rollback will not be called if *any* of the SQL statements succeed. Commit will be called if even just one sql statement succeeds. To make sure all the housekeeping is in order, ASTA will then keep the OldValuesDataSet rows that have failed on the server and delete any rows who's SQL has succeeded on the server.

1.8.4.13 TUtilInfoType

Unit
AstaStringLine

Declaration
```
type TUtilInfoType = (uiUserCount, uiMetaData, uiSQLSelect, uiSQLExec,
```

```
uiServerPerformance, uiSessionInfo, uiServerVersion,
uiAutoClientUpgradeInfo, uiWriteAutoClientUpgradeDataSet,
uiSaveClientUpgradeExe, uiPublishedDirectories,
uiWritePublishedDirectories, uiRegisterSwitchBoard);
```

Description
These surface new calls to return server information.

uiUserCount

uiMetaData

uiSQLSelect

uiSQLExec

uiServerPerformance

uiSessionInfo

uiServerVersion

uiAutoClientUpgradeInfo

uiWriteAutoClientUpgradeDataSet

uiSaveClientUpgradeExe

uiPublishedDirectories

uiWritePublishedDirectories

uiRegisterSwitchBoard

1.9 ASTA Vision

An overview of ASTA and our vision.

ASTA was designed to ease the transition to multi-tier development. Our powerful hybrid components allow developers to become successful leveraging the skills they already have. At the same time, the components are flexible enough to grow with your skills and needs. No matter whether you want your application's business logic to reside on the client or on the server side (or both) ASTA allows you to succeed! Our end-to-end component suite moves corporate data as easily as the Web moves text and graphics.

ASTA provides the ideal environment for developing business to business applications. Its extremely thin client architecture, coupled with our embedded application server technology makes it a breeze to develop internet enabled applications quickly and reliably – i.e. we have already done the hard work for you by building a middle-tier that can be used straight out of the box. Or if you prefer, you can use our embedded

application servers as a framework within which you can plug your business logic and employ easy-to-use data-aware controls on the client side – true RAD multi-tier development.
ASTA dramatically reduces an application's life-cycle costs - resulting in faster development, easier maintenance and near-zero administration.

1.9.1 N Tier

Before describing ASTA itself, it may be useful to provide an indication of exactly what we mean when we talk about n-tier (or multi-tier) applications.

An n-tier application is simply one which has the capacity to run on multiple physical machines and/or within multiple separate processes on the same machine. A process might be an executable application or a library (such as an ActiveX or Windows DLL) that has the capacity to share it's resources and functionality.

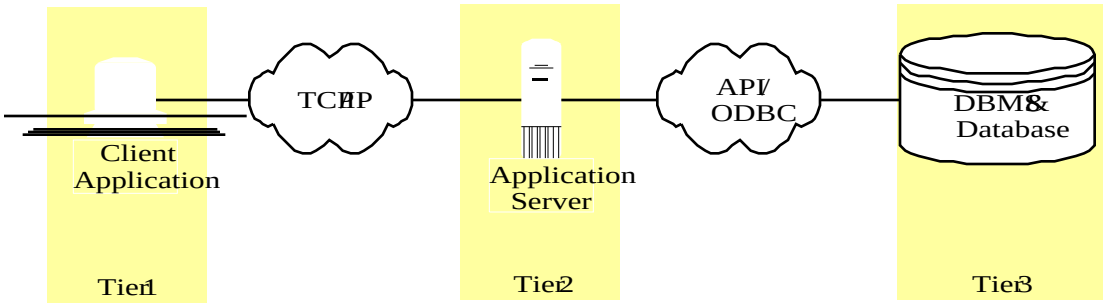
The precursor to n-tier is the client/server application where there is a delineation between the client (as seen by the user) and the database server located probably on a server machine on a network. These applications are commonly called "fat client", as most of the business logic for the application lies on the client side.

The evolution of n-tier was spurned by the realisation that performance, scalability, deployment, sharing of functionality and application maintenance could be significantly enhanced through the further delineation of application functionality into any number of separate processes (multiple tiers). This allowed applications to be developed where each key part of the business logic could be encapsulated in it's own process. This provided the ability to deliver real "thin clients", as most of the applications functionality could be stripped out of the client to be located appropriately on a server or other machine visible to the client.

Thus, the 'n' in n-tier simply denotes an undefined number of tiers.

The most common manifestation of the n-tier paradigm is the 3-tier model, an example of which is shown in Figure 1. This usually entails the following logic:

- User interface functionality is located in the client application.
- The majority of business logic is located in the middle tier (sometimes called an application server).
- The data is located within a database which is managed by a DBMS.



A basic 3-tier structure.

Whilst the above example is a very simple explanation of the paradigm, it is adequate to explain the concept – in reality, some program functionality may be shared between tiers.

So what is happening in each of these three tiers? Firstly, the client application manages the user's interaction with your system. It includes all of the screens and dialogs that allow the user to interact with your program. Secondly, the application server (or middle tier) includes the actual business logic of your program – this tier provides the main functionality of your application. The middle tier (as it's name suggests) manages and controls all interaction between your users and the database tier. Finally, the DBMS/database tier includes your actual data and the processes required to manage that data.

The middle tier is the part of your application which makes it possible to abstract your client applications to remove any dependencies on a specific database back end. This arrangement provides a sound basis for the n-tier computing model.
This is the model which is most commonly used within ASTA applications where the middle tier is referred to as the *ASTA server*.

1.9.2 In a Nutshell

What is the ASTA vision in a nutshell? It is to support application development that spans a wide range of platforms, usage models, programmer skill sets, languages, applications, appliances and hardware devices etc. It's about multi-tier application development, usually involving database access.

Software development is moving rapidly toward an n-tier model, and we are providing the support needed to assist developers in harnessing the best of this model – now and into the future.

ASTA has been designed, written and is supported by developers vastly experienced in both database and multi-tier development. Our aim is to partner you in this "brave new world". We can help by ensuring developers don't have to learn new techniques - they can leverage their existing skills, and we will lead them with little pain to the new technologies now and in the future.

ASTA has developed into a mature set of components which provide a strong base for the incorporation of relevant new technologies. This claim is supported by our integration of a number of technologies into the ASTA component set, including: XML, JDBC, Java classes, Palm classes and WinCE classes – with more to follow.
- -- --- -- -
One of the big pushes in the n-tier model has been the move toward browser-based client applications. ASTA believes that whilst some applications will benefit from this approach, there are too many inherent limitations for the majority of database applications.
One benefit that n-tier brings (as demonstrated by the internet) is the potential for processes to be run not only on different machines, but on different platforms using a variety of communication methods and hardware devices. ASTA currently provides support for development on Win32 platforms and (with the release of Kylix) adds support for Linux based ASTA applications. We have also developed ASTA messaging classes for use with Java, Palm and WinCE, allowing the expansion of ASTA client side development to other areas like wireless and handheld devices. Other connection options include dedicated ASTA JDBC and ODBC drivers which open up ASTA development and platform

options even further.
- -- --- -- -
Provision of the right tools isn't the whole picture, ASTA also recognises the importance of support and developer relations. To this end we have built up one of the most comprehensive support networks in the industry, providing unparalleled technical support to our users. This is enhanced by our newsgroup and eGroup networks which promote an "open help" environment, giving all of our users the opportunity to interact directly with the ASTA development team.
- -- --- -- -
Distributed networks are changing the world so let us be your long term distributed technology partner. Whatever you need now and tomorrow, chances are that we have already thought about it.
But most importantly, ASTA continues to embrace new technologies, incorporating the best of breed into the existing ASTA tool set. This will ensure that developers can easily expand their ASTA applications to cope with this changing world with little fuss and minimal learning.

1.9.3 Features

Two key benefits of ASTA's design are that:

1. It abstracts the database application development process so that any database components can be used on ASTA Servers
2. It handles all threading internally so that developers need not be concerned with threading issues.

1.9.3.1 Flexibility

ASTA technology is especially well suited to Business to Business (B2B) application development. Moreover, with ASTA you can implement your application once then deploy it on a standalone PC, in a LAN as well as over the Internet. With ASTA, you can develop ultra thin client applications which run under WinCE, Palm and Java – opening up many possibilities for the development of internet-enabled applications on a variety of handheld and other appliances. ASTA lets you do multi-tier programming in many ways – it is the most flexible way to develop applications - right now

1.9.3.2 Ease of Use

ASTA empowers developers to work quickly. The ASTA component set hides the complexities of working with sockets. In addition, threading issues for concurrent database connections are handled automatically by ASTA. ASTA was originally designed to allow database developers to move into n-tier development without needing to understand anything about sockets, inter-process communication and database threading.
Our embedded application servers will allow you to get your internet-enabled applications up and running without any coding on the middle-tier if you wish. Of course, if you wish to delve further, you can dig deeper into the more advanced features of the ASTA component set.

1.9.3.3 Scalibility

Scalability is a significant issue with n-tier systems. ASTA provides three server threading models to ensure that precious server resources are properly managed. These threading models provide optimum resource usage based on expected client utilisation levels. ASTA servers can automatically

expand available connection resources without user intervention to cover peak usage requirements, and can allow changes in threading model without any need for programmer intervention.
Of course, with ASTA's embedded application server technology – all of this is available without the need for any server side coding.

1.9.3.4 Deployment

ASTA applications are easy to deploy (with no DLLs or other runtime requirements beyond the application executables). ASTA uses TCP/IP for the transport of messages between tiers, requiring no additional communication protocol and no component registration. There is no need for COM/DCOM or other reliant technologies.
ASTA also provides the ability to automatically update client applications to the most recent version. The server keeps a log of the most recent version of each client application. At login, if the server detects that the client is running an older version of the client application, then the server will stream the latest version back to the client. Each ASTA server can provide this functionality for any number of client applications.
The ASTA Delta Patch Manager allows updates to be confined to the portion of the client executable that has actually changed, drastically reducing the time taken to download and apply updates.

1.9.3.5 Platform Support

Platform Support
ASTA commenced life as a set of components for the creation of Windows applications using Delphi and C++ Builder. This has been followed by the porting of ASTA to Kylix, allowing ASTA applications to be run under Linux.
Dedicated classes are available for the development of ASTA client applications using Java, Palm and WinCE, as well as a full JDBC implementation. In addition, ASTA includes the ability to read / write data in XML format. Also in development is an ASTA ODBC driver.

1.9.3.6 Database Support

Database Support
ASTA works with the BDE and with nearly all non-BDE data access software. For Delphi and C++ Builder developers, your existing applications can be converted to ASTA with the help of the ASTA Conversion Wizard - no need to wait, you can start coding with ASTA right now - you will be ready no matter how your application needs to be deployed.
ASTA already supports a large number of DBMSs, with an ever increasing number of database connection components. This provides maximum flexibility for developers to leverage the database skills they already have. We will continue to add more database and component options as they become available. In addition, future versions of ASTA will support 'pluggable database component architecture', allowing developers to easily swap database components within an ASTA server.
In addition, ASTA's dataset components descend from the standard Delphi dataset components, meaning that your ASTA enabled applications can make use of all the same data aware controls that you are already using

1.9.3.7 Messaging

ASTA provides a complete messaging infrastructure which supports both synchronous and asynchronous communication between tiers. This messaging compliments the database access as well as providing a complete non-database messaging layer. Our aim here is to provide the developer with the most complete set of development tools for inter-process communication – based mainly (but not exclusively) around database enabled applications.
In addition, ASTA allows the same core messaging calls to be used across all platforms and thus within all appliances and hardware devices that we support.

1.9.3.8 Security

ASTA provides security for your applications and data in a number of ways.
Firstly, there is a fully configurable login sequence between ASTA clients and servers. The developer has full control over what happens following a login attempt (successful or not) by a client.
Secondly, developers can seamlessly plug in their own encryption and compression routines, allowing all information to be encrypted and/or compressed before transfer between tiers.
For controlling access to visual components and functionality within your applications, Tools&Comps have created a special ASTA version of their component level security product.
For industrial strength security, ASTA provides ASTA Secure Sockets. This is an enterprise wide solution, designed for high performance encryption and access control to a bank of ASTA servers. It uses certificate based encryption, unique session keys, firewalling, message switching and logging. Thus, it can provide unparalleled protection for your ASTA servers, and is secure enough for the most demanding applications (eg. e-commerce or medical or police records

1.9.4 ASTA vs the Browser

A discussion of the appropriate use of browser technology in complex business applications.

The World Wide Web, and the servers and browsers that make it work, are fantastic and highly useful pieces of technology. But like all tools, the web has appropriate and inappropriate uses. Whether you entrust your business to a technology like the web or a technology like ASTA is a question of appropriate use.
We believe that the classic web architecture - a light browser, a protocol like HTTP, and a (database-connected, if necessary) server - is the best way to handle many lightweight applications.

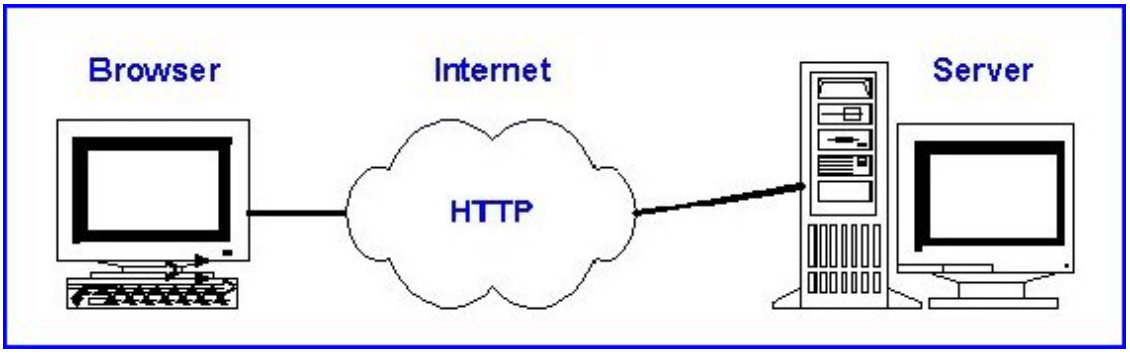


Figure 2: Connection via a browser

A website is an example of an appropriate use of this technology - the web is great for presenting read only data. Even though the technology created by ASTA can perform this task, it is handled well by the WWW and is the better choice for publishing (relatively) static pages of information. But while the WWW is good for simple data, we feel that it is a poor choice for implementing complex applications, even with Java or ActiveX. The web seems particularly ill-suited for data driven, enterprise wide, business applications.

ASTA clients can, however, *appear* as HTTP to be able to get through firewalls.

1.9.5 What's wrong with the Web?

The problem with the web is fundamental to it's architecture and manifests itself in two places, the protocol and the interface:

- HTTP, the language between servers and browsers, is a stateless protocol
- HTML has a limited interface and limited utility
- Browsers cannot work disconnected

A "stateless protocol" is one where the connection is not maintained. Put simply, the web can't "push". Clients can ask the server for information, but the server cannot send information to the clients that clients did not specifically ask for. For instance, a customer might trade stocks using a browser on the Internet. But the brokerage notifies the customer that their stock trade has occurred by sending email. This is the result of an architecture built on a stateless protocol, the server cannot update the browser. While this is not a critical problem for a casual surfer, it is unacceptable in a business environment. We do understand that in some circumstances that this cannot be avoided and ASTA can function just fine as ASTA supports stateless or disconnected messaging to be able to simulate server push with stateless clients.

The second problem is the limitations of the Hyper Text Markup Language (HTML). At first, the web seems to have a rich interface - look at all the rich text, images, video and even audio! But the fact is that those same media can be readily delivered in business applications. They simply haven't been. Why? Probably because most businesses are more concerned with basic commerce than entertainment. Hopefully, as a result of the web, "regular" applications will become more lively without sacrificing their usefulness. But despite the varied media riches, HTML is not practical for business tasks. HTML has inadequate database support and interface tools.

The third problem, which the popularity of PDA's like the Palm and IPAQ's have showed us is that browsers cannot work disconnected. In the new Wireless world, client must be able to work disconnected as well as connected. Connect to a server, get some data, disconnect and go about your business. No longer, is the world controlled by Desktop PC's only connected to the internet.

1.9.6 Web Burn

What is web burn? Web burn is our name for the phenomenon whereby well-intentioned teams set out to deploy a web based solution and they "fail". They typically produce a useful site, but one that is far short of intended use and functionality. Furthermore, the effort always costs much more and takes much longer than anticipated. There are good reasons for this! The web wasn't designed for complex applications - that's not an appropriate use. The web is great for simple applications, but it is terrible at producing complex business applications. The promise of simplicity is born in the effortless assembly of that first web page, but the promise dies as you try to implement "real" functionality.

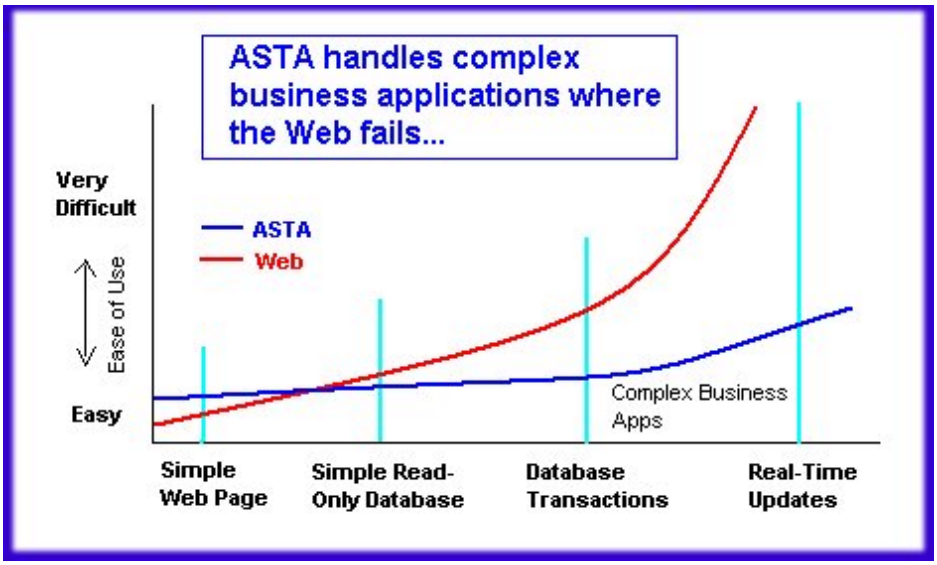


Figure 3: Application Complexity versus Ease of Use

As you start to add more functionality, you become overwhelmed with complexity and new technologies. ActiveX, VBScript, DHTML, JavaScript, Java, CGI, ISAPI, etc. As you try to add functionality, you find the implementation curve steepens rapidly. The problem is compounded by the bleeding edge factor of these new technologies.
You might argue that the web can "push" and that you can use rich controls thanks to ActiveX and Java. But if you look at the matter objectively, it becomes apparent that ActiveX and Java are attempts to "kludge" the web. Attempts to get it to do something it wasn't designed to do - the proverbial square peg in the round hole. The great irony is, that in the rush to "fix" the web some of it's best traits are being destroyed. How lightweight is a 15 MB browser? How far off is the 50 MB browser? Instead of the simple, lightweight browser that once inspired us with awe, we have memory hungry, resource eating browsers. ActiveX brings it's own layer of complexity to the application. Juggling ActiveX around the network has its own problems.
Java too has its special features. But the Java Virtual Machine or JVM seems like an odd thing to us. Nobody has ever asked us if we could please make their machines run slower or their interface seem less responsive. We don't expect that to change. Furthermore, these are rapidly emerging and therefore changing or "unstable" technologies. Each operating system upgrade (maybe even each financial quarter) brings upheaval. Working with the technology is a constant wait; you end up waiting for the features that will finally make the product do what it was reported to do two years ago! And how about those browser wars? Nothing like mixing a little market turmoil in with your technology just to keep things interesting

1.9.7 Why ASTA succeeds where the Web Fails

ASTA was designed from the ground up to be a superior business solution. This isn't an academic solution, it's a practical one.

ASTA was inspired by the web. We took the best features of the web and balanced them against the needs of business users. We also examined the bad things about Java, ActiveX and emerging technology in general. ASTA is the hybrid result of our studies and labour. ASTA takes the best features of the web, adds the capabilities that are critical to businesses, and purposefully avoids the leading bleeding edge pitfalls of other technologies.

ASTA is easy
ASTA produces complex applications with ease. It was designed to! ASTA handles datasets as gracefully as the web handles text and graphics. ASTA was also designed to "push" data to clients and to deliver real time messages.

ASTA is a practical technology
ASTA's unique properties are the result of it's unique beginnings. From the start, we decided that this technology needed to appeal to four camps; the system administrators, the application developers, the end users, and the CFO. It took the combined visions of an applications programmer, a systems professional and seasoned business veterans.

To please the end user, we delivered a native Windows application - providing a crisp and responsive graphical interface. Because it's a native Windows technology, ASTA programs can take advantage of existing operating system benefits and future benefits as well. If it can be done in Windows, it can be done in ASTA. That includes sharing data with other programs (word processors, spreadsheets, browsers). With Kylix, this applies to Linux as well.

For the applications programmer, we laboured to present the multi-tiered architecture in an easy and intuitive manner. As a result, a developer with no distributed development experience will have little trouble working with ASTA. In fact, ASTA is surprisingly well-suited to the task of migrating existing "fat client" applications to thin client, distributed client/server applications.
The systems professionals received special attention. Too many applications are dropped at their doorstep by programmers that simply don't understand the amount of effort, complexity and cost consumed by a sophisticated corporate system. They understand the programming world, but make no provisions for the applications overall impact on the system (and therefore it's unseen impact on that part of the business). What good is a $5,000 application that costs $50,000 to install and administer?

Our solution was to deliver a thin client application that can be centrally controlled; it installs from a central server, it is upgraded from a central server, and the business rules are maintained at a central server. ASTA also recognizes that the cost of bandwidth is often the highest recurring cost in the IS budget. We are miserly with bandwidth. And ASTA's ability to operate in several modes provides for unlimited flexibility when trying to preserve or provision existing bandwidth.

ASTA's chief proponent and advocate, however, might be the corporate CFO. ASTA can be connected to disparate data sources; including "legacy systems", AS/400s and mainframes - custom server interfaces can also be written. Since ASTA has such a small footprint, it will run robustly on inexpensive NetPCs. Instead of implementing resource hungry Exchange type environments, a company should consider stepping off the vicious upgrade cycle and implementing a series of lightweight ASTA applications. The graphical front end extends the life of the legacy systems and preserves the skill investments of in-house personnel. Furthermore, the "rocket science" of writing a complex, globally capable, thin client, n-tier application is encapsulated in ASTA. Programmers with average skills and experience are able to produce sophisticated applications without needing months of training and studying new technologies.

1.9.8 ASTA is not "trade journal technology"

What is trade journal technology? That's easy, trade journal technology refers to programs or applications that work only in trade journals. When you try to run them in your business on your computers, you discover that the actual solution isn't one tenth of what it was reported to be.

ASTA isn't based on emerging or evolving technologies. It works in real life, not just trade journals. ASTA was purposefully founded on proven technologies like TCP/IP - an open standard in near universal use. ASTA was intended to be simple and reliable. That is our agenda: to provide a simple and reliable technology to enable the next generation of n-tier computing.

ASTA will lower your costs and ease administration
ASTA applications can significantly reduce your TCO (Total Cost of Ownership). The clients can be installed from a central server. Once they are on the client PC they can be self-updating, upgrading themselves as newer versions are placed on the server. There are no DLLs, to juggle, no drivers to upgrade.
At about 1.0 MB, the applications have such a light footprint that they run well on low-cost NetPCs.

1.10 ASTA SkyWire

1.10.1 Welcome to the Wireless Revolution

The days of the Desktop PC's high growth rate have passed with the coming of the Internet and PDA's. A dozen years ago there was a mass migration from DOS to Windows and currently there is the same type of migration from Windows to Wireless. The first phase of this migration was to Internet enable all applications but in the late 90's the Palm showed one of the great weaknesses of browser only applications: the inability to work disconnected. The Internet migration can now be termed a Wireless migration and the brave new Wireless world has many additional bullet points that business will need to confront one way or another. Business may not require all the pieces detailed below but they must have some strategy to allow their legacy data to make an appearance in the new Wireless world. ASTA SkyWire is not one single piece of technology but a component based technology fabric that can be used to allow companies to make an easy and quick transition to take advantage of the Wireless revolution.

The ASTA Hitchhiker's Guide to the Wireless Universe provides a road map of what is needed in this brave new wireless world.

1.10.1.1 Hitchhickers Guide to the Wireless Universe

1. Run over any TCP/IP connection and be able to disguise yourself as http at anytime.
2. Have a middleware server that supports ASP and Peer to Peer.
3. Be aware that the most precious resource is bandwidth.
4. Provide XML Support.

5. Provide Compression Options out of respect for our most precious resource: bandwidth.
6. Support Web Services on both client and servers.
7. Provide Encryption Options to be able to operate securely.
8. There shall be no administration required (Zero Admin rule).
9. Authenticate at the server by username and password and be able to uniquely identify remote hardware devices.
10. Once a client app is deployed it should be able to update itself automatically and efficiently.
11. Be able to work connected and disconnected with sync technology to bridge those states.
12. Work with the most popular Development tools.
13. Work with any Database, on any Platform and provide a way to move data between any database and any device.
14. Have a component architecture so that the same techniques can be used cross platform

1.10.1.2 ASTA Unified Messaging Initiative

A World of Disparate Platforms and Systems

Today's world of computing within companies and organizations is often a hodge-podge of different platforms, operating systems, and equipment. Each system is used for a specific purpose, and is often legacy or the by-product of an acquistion or takeover. An example of a modern day organization's information systems infrastructure might be of a mainframe for reliable and scalable manufacturing software, workstations to operate the equipment and work as a system server, and a scad of networked personal computers. Usually these areas overlap, and so rather than one system, it might be a cluster or collection of systems, from different companies. The aforementioned mainframe might be a legacy Sperry mainframe with several DEC VAX minis working in tandem.

Because of the amount of software, and knowledge base for development and maintenance, moving to a new platform is not often a viable option. Nor can an organization invest the money, time, and skill to simply drop its current systems. Still, software needs to be developed to get these different platforms to talk to each other, and even more difficult, work with newer devices. It would not be unusual for a company to want to give its warehouse clerks handheld or palmtop units to enter and track inventory, which is then maintained on a legacy minicomputer or mainframe. A more ambitious project would be to interface this legacy system to the web, and allow for warehouse management through a web page.

In each case there are a myriad of different techonologies, protocols, and standards available to a companies IT staff. The major problem is to find a solution that will work with legacy systems, do what is needed now to accomplish a task, and yet still provide scalability and flexibility to meet future needs. An IT manager has to accommodate the past, accomplish a current project goal in the present, and plan for the future.
ASTA Technology Group has a unified messaging model that is a way to accommodate the need for platform inter-operability, but still be workable in the future. The software ASTA has allows for quick development of client-server software, using ASTA's custom message format. It is the ASTA message format that allows for flexible and powerful platform inter-operability.

ASTA Technology Parameter Message

The ASTA message format is centered on the two concepts of a message parameter, and a list of message parameters. These concepts are simple and orthogonal, so provide a simple yet powerful means for a developer to write software for platform inter-operability.
The ASTA message parameter consists of 5 elements. These elements are:

Parameter name - the name of the parameter
Parameter type - the mode of the parameter, in, out, inout, result. On SkyWire clients this is not surfaced but this will be used on SkyWire Scripting and SOAP servers where remote SOAP or ASTA SkyWire methods will be able to be define using VBScript or COM.
Data type - the type of the data, which follows the database types (char, bit, integer, etc.)
IsNull flag - a flag indicating if the data is null (a bit representing true or false)
Bytes of data - the actual data as a list of bytes

The ASTA parameter message follows very closely the concepts of data in a relational database. This builds upon existing knowledge, and creates a message parameter structure that is highly compatible with a database system. The ASTA message parameter provides a logical consistency by focusing on the data, and not on how the message parameter will be transported.

The ASTA message parameter list is a collection or list of the ASTA message parameters. The API to process a parameter list is organized around the data types. For each type, the individual ASTA message parameter can be referred to by the position in the list or index, or by the parameter name. Using the parameter name is similar to the associative arrays found in PERL, and is immensely powerful. Manipulating and handling ASTA message parameters within a parameter list is simple, following the data types, and powerful as each message parameter can be accessed by name or index.

ASTA provides a logically consistent and powerful yet easily used mechanism to construct, manipulate, and handle a message.The actual means of transporting the ASTA parameter message list is to convert the information to and from a string type. This is transparent, and handled by the different implemented libraries and packages for different platforms. The format for transport is the same at the lower "at the wire" level, but is conceptually the same in each language implemented, although the actual data representation may differ. ASTA parameter message lists can be implemented efficiently in a native representation in a programming language on a platform, but all can be transported to the different platforms. Hence ASTA's messaging system is natively platform efficient yet inter-operable.

ASTA, with its message parameter so closely allied with a database representation of information, provides a means for the seamless transport of data. Not data translation, but transportation. In doing so, ASTA provides the means for seamless platform inter-operability and data exchange, whether that platform is a legacy mainframe, or new state of the art handheld.
XML and SOAP Technology. One of the latest developments is the rise and surge of interest in the eXtensible Markup Language (XML). The power of XML is in its ability to allow for application independent data representation that follows a consistent pattern. XML allows for custom data representation that can be exchanged among different apps, and transformed.

ASTA's message structure can be converted to and from an XML form. Using the wide assortment of available XML tools, components, and libraries, the XML can be processed

as needed by an application. XML is another form that an ASTA message can be transformed to, then allowing an ASTA message to be used and moved through XML-oriented applications.
Another offshoot of XML is the development of the Simple Object Access Protocol or SOAP. Using SOAP, components as objects on a system can be instantiated, data marshalled, and methods called and results returned. ASTA supports this, by allowing the ASTA message to be packed into a SOAP message and delivered to a SOAP-compliant system.

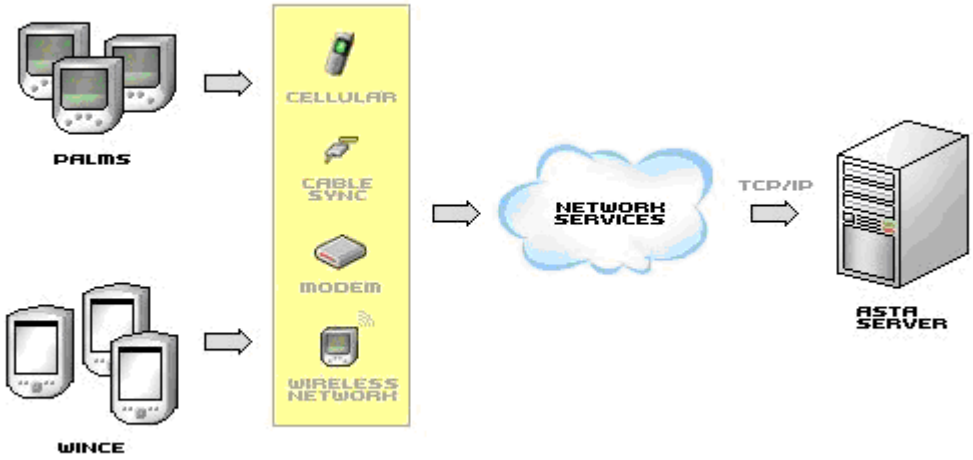
With ASTA, a collection of heterogenous systems and applications can inter-operate and exchange information seamlessly. A typical application might be one where users with a wireless handheld enter data, such as during an inventory of parts, submitting an expense report, etc. This data is sent as an ASTA message to an Interbase server on a Linux system. Another user, responsible for creating a weekly summary in Access on NT, can access this database and using ODBC, move the data in and out of the Access database on their NT computer. The results can be sent to another server by HTTP using SOAP to marshall an object that performs the data update.

Hence this organization has a seamless and transparent mechanism to gather and collect data, store it in a central location, then allow it to be updated in a specific way, and then finally archived in a larger legacy system.

1.10.2 Introducing ASTA SkyWire

The ASTA SkyWire architecture consists of a SkyWire Server and one or more SkyWire thin clients. ASTA Win32 Administrative thin clients are available to help administer remote Databases through SkyWire Database Servers. Since SkyWire clients and servers are available on many platforms, they may be packaged in different formats: as shared libaries on the Palm OS, DLL's on Win32 or WinCE devices, COM DLL's on WinCE or Win32 32, so's on Linux or as Java classes or as C# XX files.

Below is a diagram demonstrating the SkyWire architecture. SkyWire clients and servers are capable performing standard tasks.



1.10.2.1 Standard Tasks

The table below represents basic server and client side functionality that must be able to be implemented according to the Hitchhiker's Guide to the Wireless Universe.

Client Communication Tasks	Server Communication Tasks
Connect to a Remote Server	Accept Connections from remote clents
Be Authenticated at the Server	Authenticate Clients
Send messages containg any kind of data	Receive messages containing any kind of data
Compression Options	Compression Options
Encryption Options	Encryption Options
Be able to get a current version of the Application	Be able to provide updated applications for client

Client Application Tasks	Server Application Tasks
Request SQL Result Sets and data structures	Provide Result sets for SQL Selects
Request Database Metadata	Provide Database metadata
Exec SQL in a transaction	Provide ExecSQL support and support for transactions
Execute Business Methods on a Remote Server	Business Logic on the Middle Tier

SkyWire uses a component based architecture to facilitate what is required for the Wireless world.

1.10.2.2 SkyWire Servers

ASTA SkyWire servers are both scalable and easy to deploy. If need be they can easily be run as primary application servers but they can also be run on normal desktop pc's with no deployment requirements except to start them up. In many cases SkyWire servers are database servers and can connect natively, on any platform to just about any database available.

SkyWire Servers can be created with Visual Basic using the SkyWire Server COM Interface, with native VCL components with Delphi or Kylix or with Java. SkyWire servers can provide database support via the SkyWire Server API or implement custom methods to allow for Business Logic to be maintained on the middle Tier.

SkyWire Servers can supply SOAP Services as standalone HTTP Servers or through ISAPI DLL's, Apache DSO's or Java Serverlets.

SkyWire servers can also can act as proxy or adapter servers to communicate with the large Application servers via XML messaging. Jasta is an ASTA Java Server that uses RMI to communicate with EJB severs and allows for all of ASTA cross platform clients to make calls to EJB Servers. ASTA XML messaging makes it easy for ASTA servers to be plugged in between proprietary corporate of mainframe data and allow the whole fabric of ASTA connectivity to them be available.

1.10.2.3 SkyWire Clients

Skywire clients are available on a variety of and operating systems using the most popular development tools. There is also an Asta thin client ODBC Driver, type 3 JDBC Driver and Win32 clients in development to execute EJB from Java Servers.

Language	Development tool	Palm	WinCE	Win32	Linux
Basic					
	AppForge	Yes	Yes		
	CASL Soft	Yes			
	eVB		Yes		
	NSBASIC	Yes			
	Satellite Forms	Yes			
	Visual Basic			Yes	
C#	Visual Studio Dot Net			Yes	
C++					
	C++ Builder			Yes	
	CodeWarrior	Yes			
	eVC++		Yes		
	Falch.net	Yes			
	gcc				Yes
	VC++			Yes	
Java		Yes	Yes	Yes	Yes
Pascal					
	Delphi			Yes	
	PocketStudio	Yes			

1.10.2.4 SkyWire Component Architecture

The success of Windows and the Desktop PC was epitomized by the Byte Magazine Cover of 1994 that celebrated the success of Visual Basic Components over the abstract and unfulfilled OOP (Object Oriented Programming). Microsoft created Visual Basic Components and allowed Visual Basic programmers, of all skill levels, to deliver Windows Applications easily. ASTA SkyWire is bringing that same type of component architecture to the Wireless world.

ASTA has committed to supporting all of the popular development tools on Palm, WinCE, Linux Pda's, RIM Blackberries, Win32 and in Java. No matter what development tool selected, no matter what operating system or hardware, ASTA Skywire will be the familiar face allowing developers to concentrate on delivering their application knowing that next week, next month when they will be required to deliver the same application on other hardware, they will be able to use the same component methods and properties.

Function	Client	COM Server	Delphi Server	Java Server
Communication	AstaConnection	CAstaCOMServer	AstaPdaServerSocket	AstaServerConnection
Data Item	AstaParamValue	CAstaParamItem	AstaParamItem	DataValue
Data Container	AstaParamList	CAstaParamList	AstaParamList	DataList
Database ResultSet	AstaDataList	CAstaRecordSet	TDataSet	JDBC

1.10.2.4.1 ASTA Binary Patcher

The AstaBinaryPatcher allows any 2 binary files to be compared and a "patch" file created that can be applied to the first binary file using a small console program (patcher.exe). A GUI program, or the AstaBinaryPatchManager.exe is used to create and maintain the patches. Then the console program (patcher.exe) is used to apply the patches.

The typical use of the AstaBinaryPatcher is to efficiently update remotely deployed EXE's without have to stream down large files. There are 3rd Party tools that integrate with the AstaBinaryPatcher to provide a transparent autoupdate process for any application and allow the application to "check" for an update on a remote server, get a patch and update an application in place.

- TMS Software TWebUpdate
- Web Update 2

Although EXE's are normally used with the AstaBinaryPatcher to allow for remote EXE's to be made current, any file, including streamed database files could be used with this technology.

The AstaBinaryPatcher is part of the Zero Administration Initiative that is part of the ASTA's Hitchhiker's Guide to the Wireless Universe which also includes update technology for Palm OS also.

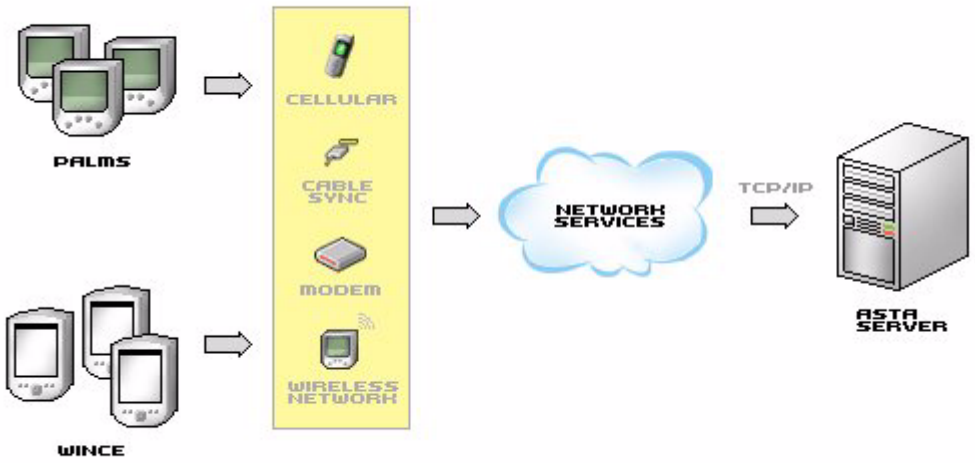
ASTA is fully committed to thin client applications with zero administration. To this end, the ASTA AutoUpdate Process allows for remote clients, when logging onto an ASTA server to have a version check performed on the server and if a more recent version of the client application is available, that version can be streamed down to remote clients, replace the existing application and relaunch the application. This process is described with examples and screen shots in the AstaAutoUpdate Tutorial. This same technique is available on PDA devices also, where a updated Palm prc or WinCE EXE is made available on ASTA SkyWire servers and then streamed down to the handheld device when a client is authenticated at the server. For low volume servers or over local lans, this works quite well but as user counts start to rise into the hundreds, it could be quite common for 100 or so users to hit a server at the same time, streaming down a 1MB + EXE to remote clients. Quite a distraction for an ASTA server designed to serve up database queries!

For high volume servers, or just for more efficient use of resources, it is quite easy to use a remote Web Server like IIS (Internet Informormation Server) to manage the most recent client application. This technique is described in the Asta HTTP Update tutorial complete with code examples and even includes a progress bar and the ability to abort the download. The ultimate solution would be to use some technology that could apply a

binary patch on the original client EXE to bring it update to date with the current version. Thus, instead of a 1MB client application a 200K patch file could be sent to clients. Combining this with the HTTP download technique you have the best of both worlds: no resources used from the ASTA server and a much smaller file streaming down to remote clients. The ASTA Binary Patcher allows any 2 binary files to be compared and a "patch" file created that can be applied to the first binary file using a small console program (patcher.exe). A GUI program, or the AstaBinaryPatchManager.exe is used to create the patches. Then the console program (patcher.exe) is used to apply the patches. Screen shots of the AstaBinaryPatch Manager are available.

1.10.2.4.2 Sky Wire Server

ASTA SkyWire, a component based architecture, consists of remote thin clients, connecting to a middle tier Application Server. SkyWire Servers are available from ASTA using a number of development tools. This document describes how to implement a SkyWire Server using Visual Basic and allow remote PDA clients to send and receive data to and from the server.



SkyWire Servers have been designed to provide the following functionality using the SkyWire Server API:
Receive Requests from Remote Clients
Authenticate Clients using a UserName and Password
Allow any kind of Data to be transmitted efficiently between server and clients
Allow clients to exec Select and ExecSQL and get server Metadata information
Provide Compression Options
Provide Encryption Options
Provide a RecordSet type of data structure so that result sets can be easily used by Remote PDAs
Allow custom servers to be easily coded to provide methods to implement server side business logic

The SkyWire COM Interface consists of the following components
CAstaCOMServer
CAstaParamList
CAstaParamItem
CAstaRecordSet

1.10.2.4.2.1 Server Components

SkyWire Servers can be built using the SkyWire COM Interface, native Delphi components, or Java. There are examples using Visual Basic and Delphi included in the SkyWire COM Server installation.

The SkyWire COM Interface consists of the following components
CAstaCOMServer
CAstaParamList
CAstaParamItem
CAstaRecordSet

Development Tool	Description
Visual Basic	This is a basic sample to show the absolute minimum required to build a server
Visual Basic with Northwind Example	This is a full database example using Northwind to show how to build a server that supports the SkyWire Server API as well as to implement Business Logic on the Server
Delphi	This is a basic sample to show the absolute minimum required to build a server

CAstaCOMServer provides a COM Interface to allow for SkyWire Servers to be created using Windows Tools such as Visual Basic or Delphi. CAstaComServer provides a "listening" ServerSocket that can receive requests from remote clients, authenticate them, and process any kind of datathrough the use of AstaParamLists. Data can be optionally compressed and/or encrypted to provide for efficient and secure communication. CAstaCOMServer is delivered with AstaCOMServer.dll which must be registered with the operating system. The ASTA SkyWire COMServer install will register the DLL or it can be registered manually by calling regsvr32.exe AstaCOMServer.dll

With the CAstaCOMServer interface, ASTA SkyWire servers can be customized to implement business logic on the Application Server or allow for remote clients to send their own SQL to the server to be processsed.

Properties
Active
AstaServerName
Compression
Encryption
IPAddress
Port
SecureServer
StringKey

Events
OnAuthenticate
OnClientConnect
OnClientDisconnect
OnParamList

Starting a Server
Handling Client Requests
Example of Middle Tier Business Logic as coded on the Server

CAstaComServer

Compression As **Boolean**

Description
Setting this property will turn on AstazLib compression which is available cross platform

Example

AstaServer.Compression = True

CAstaComServer

IPAddress As **String**

Description
Servers listen on a given network card and by default CAstACOMserver will use the first network card found. If you have multiple network cards in a computer, running an ASTA SkyWire Server, the IPAddress can be used to set the Server to listen on a specific network card's IPAddress.

Note: when running over the Internet, a static IPAddress is required and this will most probably NOT be the network card address of the computer the server is running on but the GateWay address.

Example

IPAddress = "10.0.0.101"

CAstaComServer
Function Port As Long

Description
Defines the port that any SkyWire server and start to listen from.

CAstaComServer
StringKey As String

Description

Sets the StringKey that is used for AES (Rijndael) or DES Encryption.

CAstaComServer
Function SecureServer As Boolean

Description
Sets a server to force clients to authenticate.

Example

CAstaComServer

Encryption As **EnumEncryptionType**

Currently suporrted values include:

etNoEncryption
etAESEncrypt
etDESEncrypt

Description

ASTA has implemented cross platform Encryption using the AES or Rijndael Encrytion or DES Encryption. AES is currently only avaialable to residents of US and Canada while DES, which is weaker encryption is available world wide with the exception of countries on the US watch list.

By setting this property, all data streams are encrypted between client and server.

Example

CAstaComServer

AstaServerName As **String**

Description
Since remote SkyWire clients can connect to any IPAddress and Port, the AstaServerName can be used to uniquely identify servers that may be running from the same IPAddress on a different Port.

Example

AstaServerName = "Accounting Server"

CAstaComServer

Function OnAuthenticate (UserName As **String**, Password As **String**) As **Boolean**

Description
When SecureServer is set to True then remote clients will pass in their UserName and Password to allow for Authentication to be performed. If the User is allowed to access the server, the OnAuthenticate Function should return True

Example

```
Private Function AstaServer_OnAuthenticate(ByVal Username As String, ByVal Password As String)
As Boolean

   AstaServer_OnAuthenticate = True

End Function
```

CAstaComServer

OnClientDisconnect(ByVal UserName As **String**)

Description
An event that fires when a remote client's disconnect. If the clients are required to be authenticated and provide a UserName that will be available otherwise UserName will be shown as the remote IPaddress and Port.

Example

```
Private Sub AstaServer_OnClientDisconnect(ByVal UserName As String)

   LstUserLog.AddItem "Client Disconnected at " & Format(Date, "dd/mm/yyyy") & " " & Format(Time,
"hh:mm:ss") & " UserName " & UserName

End Sub
```

See also OnClientConnect

CAstaComServer

OnParamList(ByVal MsgID As **Long**, ByVal InParamList As AstaCOMServer.ICAstaParamList, OutParamList As AstaCOMServer.ICAstaParamList, ErrorCode As **Long**, ErrorMessage As **String**)

Description
This is where all client communication comes throug, after clients have been authenticated. Remote Clients create and fill up AstaParamLists with data and send them to the server with a Msgid As Long Token that can be used to trigger a server side process. The ParamList from the remote client will come in as the InParamList and on the server, you process any incoming data and fill up the OutParamList with any response that goes to the client. If for some reason, there is a problem or error on the server, set the ErrorCode to a non-zero value and the ErrorMessage to the message you want relayed to the remote client.

SkyWire servers are available pre-compiled that respond to client requests in using a Standard API and the examples provided using Visual Basic and Delphi mirror that standard.

Error Handling

Example

```
Private Sub AstaServer_OnParamList(ByVal MsgID As Long, ByVal InParamList
As AstaCOMServer.ICAstaParamList, OutParamList As
AstaCOMServer.ICAstaParamList, ErrorCode As Long, ErrorMessage As String)

    Dim x As Integer
    Dim RowCount As Integer

    Select Case MsgID
        Case 1001
            DoServerTime OutParamList

        Case 1003
            UserDataSet.MoveFirst
            DoSelect InParamList.Params(0).AsString, OutParamList

        Case 1004
            DoTables OutParamList

        Case 1005
            For x = 0 To InParamList.Count - 1
                LogIt "SQL ExecSQL 1005: " & InParamList.Params(x).AsString
            Next
            DoExecSQL OutParamList

        Case 1006
            DoTableFields OutParamList

        Case 1010
            If InParamList.Count > 0 Then
                LogIt "SQL Request: " & InParamList.Params(0).AsString
            End If
            DoMultiRowSQL 0, InParamList.Params(1).AsInteger - 1,
OutParamList

        Case 1011
            For x = 0 To InParamList.Count - 1
                LogIt "DataList Select 1011: " & InParamList.Params(0).Name
& ":" & InParamList.Params(0).AsString
            Next

            RowCount = UserDataSet.RecordCount
            If InParamList.Count > 1 Then
                RowCount = InParamList.Params(1).AsInteger
            End If

            DoDataListSelect 0, RowCount, 32700, OutParamList

        Case Else
            DoServerTime OutParamList

    End Select

End Sub
```

CAstaComServer

OnClientConnect(ByVal RemoteAddress As **String,** ByVal RemotePort as **Long**)

Description
When a remote client connects to a server this will show a remote **IPAddress and Port**.

Example

```
Private Sub AstaServer_OnClientConnect(ByVal RemoteAddress,ByVal RemotePort As Long)

    LstUserLog.AddItem "Client Connected at " & Format(Date, "dd/mm/yyyy") & " " & Format(Time,
"hh:mm:ss") & " RemoteAddress " & RemotePort

End Sub
```

CAstaParamItem is the core for the ASTA messaging intitiative as implemented across many plantforms using .
Each CAstaParamItem is includes the following data properties and also has many routines to convert to other datatypes.

Name - the name of the parameter
ParamType - the mode of the parameter, in, out, inout, result.
DataType- AstaFieldType which follows the database types (char, bit, integer, etc.)
IsNull flag - a flag indicating if the data is null (a bit representing true or false)

AsBoolean
AsCurrency
AsDate
AsDateTime
AsFloat
AsInteger
AsMemo
AsSmallInt
AsTime
AsVariant
AsWord
DataType
GetDataSize
IsBlob
IsInput
IsNull
IsOutput
LoadFromFile
Name
ParamType
SetBlobData
Size
Text

CAstaParamItem

Description

Example

CAstaParamItem
LoadFromFile(FileName As **String**)

Description
Loads a FileName into a CAstaParamItem which will have DataType of ftBlob.

Example

LoadFromFile("c:\autoexec.bat")

CAstaParamItem
EnumFieldtype defines the possible DataTypes that can be created for CAstaParamItem's

ftInteger
ftBoolean
ftFloat
ftString
ftDateTime
ftBlob

CAstaParamItem
Function AsTime As DateTime

Description

Example

CAstaParamItem

Description

Example

CAstaParamItem
Function AsBoolean As Boolean

Description

Example

IsInput

CAstaParamItem
Function IsInput ( ) As Boolean

Description
Returns whether the ParamType is ptInput or ptInputOutput

Example

CAstaParamItem
Function AsFloat As Float

Description

Example

CAstaParamItem
Function IsOutput As Boolean

Description

Returns whether theParamType is ptOutput,ptResult or ptInputOutput

CAstaParamItem

Description

Example

CAstaParamItem

ptUnKnown
ptInput

ptOutput
ptInputOutput
ptResult

Description
These are the types of parameters that can be stored in AstaParamLists. For basic messaging, ptInput is used as the default value but these values will be used when publishing middle tier business methods as SOAP Services.

CAstaParamItem

Name(Value As **String**)

Description
Sets the Name property of a CAstaParamItem

Example

Name = "Server Time"

CAstaParamItem

Function AsSmallInt As Long

Description

Example

CAstaParamItem

Function AsInteger As Long

Description

Example

CAstaParamItem

Function AsVariant As **Variant**

Description

Example

CAstaParamItem

DataType as EnumFieldType

Description
These are the types of data that can be stored in AstaParamLists

CAstaParamItem
IsNull(Value As Boolean)

Description
Sets the NULL Flag for any CAstaParamItem

Example

IsNull = True

CAstaParamItem

Description

Example

CAstaParamItem
Function IsBlob As Boolean

Description

Example

CAstaParamItem
Function AsWord As Long

Description

Example

CAstaParamItem
Function AsCurrency As Float

Description

Example

CAstaParamItem

Function AsMemo As String

Description

Example

CAstaParamItem

Function AsDate As DateTime

Description

Example

CAstaParamItem

Function AsDateTime As DateTime

Description

Example

The CAstaRecordSet represents a very fast and light "In Memory" RecordSet that can be used on the server to consolidate data that can be streamed to remote PDA's using one of 2 RecordSet DataStructures available on remote Pda clients: AstaNestedParamLists and DataLists. Pre compiled SkyWire Database Servers support client side SQL requests and with CAstaCOMServer customized server side methods can easily be coded to implement Business rules on the server. The Visual Basic Northwind server has an example of how to move data from a VB RecordSet to a CAstaRecordSet

AddField
AddNew
BOF
CancelUpdate
Close

DataSetToDataList
DataSetToParamList
Delete
Edit
EOF
FieldCount
FieldIndex
FieldName
FieldValue
Find
Open
MoveFirst
MoveLast
MoveNext
MovePrevious
Position
RecordCount
RecordToParamList

CAstaRecordSet
FieldValue(FieldName As String, Value as Variant)

Description

Sets a Field's Data to Value.

Example

CAstaRecordSet
Edit ()

Description
Puts a CAstaRecordSet in Edit mode so that FieldValues can be changed. To post records to the RecordSet call Update or call CancelUpdate to cancel any Edit or AddNew call.

Example

```
Edit
FieldValue("Name") = "Marcio Alexandroni"
FieldValue("Address") = "Sao Paulo, Brazil"
FieldValue("Value") = 1234.56
Update
```

CAstaRecordSet
CancelUpdate ( )

Description
Cancels a call to AddNew or Edit

Example

CAstaRecordSet
AddField(Name As **String**, FieldType as DataType, StringLength As **Long**)

Description
Defines a field for the RecordSet where Name is the FieldName, DataType represents the kind of data and Length represents the length if the DataType is a String Field (ftString)

Example

```
with UserDataSet
        .AddField("Name", ftString,40
        .AddField("Value",ftString,0
```

CAstaRecordSet

Function FieldName(Index as Long) as String

Description
Returns the FieldName at Index using a zero based indexing scheme.

Fields run from 0 to FieldCount-1

Example

CAstaRecordSetDataSetToParamList(ByVal StartRecord As Long, ByVal EndRecord as Long, ByVal MaxLength AsLong,
ParamList AS CAstaParamList)

Description
Transfers a CAstaRecordSet to a CAstaParamList so that it can be received and used on the client as an AstaNestedParamList and supported with the SkyWire Server API as PdaServer

StartRecord The Row to Start At
EndRecord The Row to End At
MaxLength The maximum size of the datastream, important for Palm OS which only allows for 32K data segments. If set to -1 then the whole DataSet will be sreamed

Example

```
UserDataSet.DataSetToParamList 1 5,-1,P
```

CAstaRecordSet

Update()

Description
Posts a record to a CAstaRecordset after AddNew or Edit

Example

```
        AddNew
        FieldValue("Name") = "Marcio Alexandroni"
        FieldValue("Address") = "Sao Paulo, Brazil"
        FieldValue("Value") = 1234.56
        Update
```

CAstaRecordSet

Open()

Description
Opens the RecordSet

Example

CAstaRecordSet
MoveLast

Description
Moves to the last row of the CAstaRecordSet

Example

CAstaRecordSet
MoveFirst

Description

Moves to the first row of the CAstaRecordSet

Example

CAstaRecordSet
MovePrevious ( )

Description

Moves to the previous row.

Example

CAstaRecordSet

Function Find(ByVal FieldName as **String** , ByVal Value as Variant,
ByVal CaseSenstive as Boolean,ByVal PartalKey As Boolean) As Boolean

Description

Attempts to find the first instance of Value in FieldName. By default is not case senstive and if True will position the cursor on the row where the values is found.

Example

CAstaRecordSet

RecordToParamList(Value As CAstaParamList)
Description
Transfers the one row to a ParamList that can be read on the client as an AstaNestedparamList.

See also

DataSetToParamList
RecordSets on PDA's

Example

CAstaRecordSet
Close ()

Description
Closes the RecordSet that has been opened.

Example

CAstaRecordSet
Delete ( )

Description

Deleletes any row from a CAstaRecordSet

Example

CAstaRecordSet
DataSetToDataList(ByVal StartRecord as Long, ByVal EndRecord As Long, ByVal MaxLength As Long, ParamList As CAstaParamList)

Description
Transfers a CAstaRecordSet to an AstaDataList for use by remote Pda clients.

Example

CAstaRecordSet
MoveNext ( )

Description
Moves to the NextRecord of the CAstaRecordSet

Example

CAstaRecordSet
Function FieldCount As **Long**

Description
Returns the number of Fields as defined by AddField.

Example

CAstaRecordSet
Function Recordcount as Long

Description

Returns the number of rows in the RecordSet.

Example

CAstaRecordSet
Function Position as Long

Description

Returns the position of the current row.

Example

CAstaRecordSet
FieldIndex(Index as Long) as Variant

Description
Returns the value the field at Index as a Variant.

Example

CAstaRecordSet
Function EOF As Boolean

Description

Returns true if End of File.

Example

The ASTA Unified Messaging Initiative allows for cross platform messaging by using AstaParamLists which are easy to use and fully streamable Data containers that can be transported between ASTA clients and servers and between ASTA clients regardless of hardware or operating system. AstaParamLists are implemented cross platform so that ASTA Pda Clients can send and retrieve any kind of data from SkyWire Servers. DataTransfer occurs by a client creating an AstaParamList and sending it off to a SkyWire Server where the server responds to the request, using any of the incoming Param Data, and then returns an AstaParamList back to the client.

All datatypes are suported and there is also support to add RecordSets to ParamLists so that remote clients can receive "result sets" in a row/column data structure.

Add
AddOneParamItem
AddParam
Clear
Count
CopyList

CreateParam
Count
FastAdd
FindParam
ParamByName
Params

CAstaParamList

AddParam(Name As **String**, Value as **Variant**)

Description
Adds a ParamName and Value as Variant.

Example
P.AddParam "1","Hello World"

CAstaParamList

Function CopyList(Value As CAstaParamList) As CAstaParamList

Description
Makes a copy of the CAstaParamList and returns it a result.

CAstaParamList

Function ParamByName(Value As **String**) As CAstaParamList

Description
Returns a CAstaParamList if Value exists in the List equal to the Name.

CAstaParamList

CreateParam(ByVal Fieldtype As DataType

Description
Creates a CAstaParamItem.

Example

CAstaParamList

FastAdd(Value as **Variant**)

Description
Adds a Value as an CAstaParamItem automatically creating a default Name that is unique.

Example
P.AddParam "Hello World"

CAstaRecordSet
AddNew()

Description
This adds a new row to the CAstaRecordSet. Update should then be called to Post the row. If you don't want to post the row, call CancelUpdate.

Example

```
AddNew
FieldValue("Name") = "Marcio Alexandroni"
FieldValue("Address") = "Sao Paulo, Brazil"
FieldValue("Value") = 1234.56
Update
```

CAstaParamList
Function FindParam(Value As **String**) As CAstaParamList

Description
This is a test that will return a CAstaParamList if Value exists in the List equal to the Name.

Example

CAstaParamList
Clear ()

Description
Deletes any CAstaParamItems in the List setting the count to zero.

CAstaParamList
Function Add As CAstaParamItem

Description
Adds a CAstaParamItem and returns an instance of it.

CAstaParamList
AddOneParamItem(Value As CAstaParamItem)

Description

Adds a CAstaParamItem that has already been created.

CAstaParamList
Function Count() as Long

Description
Returns the number of CAstaParamItems in the List.

Example

CAstaParamList
Function Params(Value As Log) As CAstaParamItem

Description
Returns a CAstaParamItem at index Value. Valid values run form 0 to Count-1

Example

CAstaRecordSet
Function BOF as Boolean

Description
Returns whether the record set is at the Begining of Record Set (Begin of File)

Example

There is a full example server using Visual Basic and connects to the Northwind sample database included in the SkyWire COM Server install. This server shows how you can extend SkyWire servers with full database support and also has examples of custom methods that implement business logic on the server.

Starting the Server

A server must listen on a certain port and be set to active and also connect to a database.

```
Set AstaServer = New CAstaCOMServer
Set Conn = New Connection
Set Tbl = New Recordset

Conn.Open "ODBC;DATABASE=NorthWind;UID=;PWD=;DSN=NorthWind;"
AstaServer.Port = 9090
AstaServer.SecureServer = True
AstaServer.Active = True
```

Authentication a Username and password is presented from remote clients.

```
Private Function AstaServer_OnAuthenticate(ByVal Username As String, ByVal
Password As String) As Boolean
    AstaServer_OnAuthenticate = True
End Function
```

Converting a RecordSet to a CAstaRecordSet

```
Private Sub DoDataListSelect(SQL As String, StartRecord As Integer, RowCount
As Integer, MaxLength As Integer, P As AstaCOMServer.ICAstaParamList)

    Dim RS As Recordset
    Dim AstaRS As CAstaRecordSet
    Dim x As Integer

    Set RS = New Recordset
    Set AstaRS = New CAstaRecordSet

    If Len(SQL) = 0 Then
        LogIt "   1011 Error - No SQL Specified"
        P.AddParam "Error", "No SQL Specified"
        Exit Sub
    End If

    On Error GoTo DoDataListSelectErrorHandler
    RS.Open SQL, Conn

    P.Clear

    ' Add Fields
    For x = 0 To RS.Fields.Count - 1
     LogIt " fields " & RS.Fields(x).Name & RS.Fields(x).Type
    Select Case RS.Fields(x).Type
            Case adBSTR, adChar, adVarChar, adWChar, 202
              AstaRS.AddField RS.Fields(x).Name, ftString,
RS.Fields(x).DefinedSize

            Case 203
                AstaRS.AddField RS.Fields(x).Name, ftString, 500

            Case adCurrency, adDecimal, adDouble, adNumeric
                AstaRS.AddField RS.Fields(x).Name, ftFloat, 0

            Case adDate, adDBDate, adDBTime, adbTimeStamp, 135
                AstaRS.AddField RS.Fields(x).Name, ftDateTime, 0

            Case adInteger, adSingle, adSmallInt
                AstaRS.AddField RS.Fields(x).Name, ftInteger, 0

            Case Else
                AstaRS.AddField RS.Fields(x).Name, ftVariant, 0
        End Select
    Next

    ' Data
    RS.MoveFirst
    RS.Move StartRecord
    AstaRS.Open
    While Not RS.EOF And RowCount > 0
        AstaRS.AddNew
        For x = 0 To RS.Fields.Count - 1
            AstaRS.FieldValue(RS.Fields(x).Name) = RS.Fields(x).Value
        Next
        AstaRS.Update

        RowCount = RowCount - 1
        RS.MoveNext
    Wend
```

There is a full example server using Visual Basic included in the SkyWire COM Server install.

Starting the Server
A server must listen on a certain port and be set to

```
Set AstaServer = New CAstaCOMServer
AstaServer.Port = 9090
AstaServer.AllowAnonymousPDAs = True
AstaServer.Active = True
```

Authentication a Username and password is presented from remote clients.

```
Private Function AstaServer_OnAuthenticate(ByVal Username As String, ByVal
Password As String) As Boolean
    AstaServer_OnAuthenticate = True
End Function
```

Handling Client Requests
Client requests come in using a MsgToken that supports the SkyWireServer API or allows for some custom Business Logic to be implemented on the middle tier.

```
Private Sub AstaServer_OnParamList(ByVal MsgID As Long, ByVal InParamList As
AstaCOMServer.ICAstaParamList, OutParamList As
AstaCOMServer.ICAstaParamList, ErrorCode As Long, ErrorMessage As String)

    Dim x As Integer
    Dim RowCount As Integer

    Select Case MsgID
        Case 1001
            DoServerTime OutParamList

        Case 1003
            UserDataSet.MoveFirst
            DoSelect InParamList.Params(0).AsString, OutParamList

        Case 1004
            DoTables OutParamList

        Case 1005
            For x = 0 To InParamList.Count - 1
                LogIt "SQL ExecSQL 1005: " & InParamList.Params(x).AsString
            Next
            DoExecSQL OutParamList

        Case 1006
            DoTableFields OutParamList


        Case Else
            DoServerTime OutParamList

    End Select

End Sub
```

ASTA SkyWire Servers have been avaialble since 1998 using Delphi. The TAstaPdaServerSocket is a native Delphi VCL control used to create SkyWire servers with pascal.

The TAstaPdaServerSocket descends from the TAstaServerSocket and adds the ability to communicate with remote PDA clients such as Palm, Wince, Linux and Rim Blackberry clients. The TAstaPdaServerSocket provides the basis for the ASTA SkyWire technology.

The AstaPdaServerSocket implements AstaParamList messaging to communicate with remote PDA Clients and provides Authentication, Encryption, Compression, AutoUpdate and Device Validation as well as routines to convert server side TDataSets to ASTA Pda RecordSet DataStructures like AstaNestedParamLists and AstaDataLists.

Properties
AllowAnonymousPDAs

Methods
PdaSendParamList

Events
OnPalmUpdateRequest
OnPDAAcquireRegToken
OnPDAAuthentication
OnPDAGetFileRequest
OnPDAParamList
OnPDAPasswordNeeded
OnPDAValidatePassword
OnPDAValidateRegToken

Security Issues

TAstaPdaServerSocket

The AstaPdaServerSocket enforces security a number of ways.

No Security
To allow for remote PDA clients to connect to a server with no authentication or validation

1. Set the AllowAnonymousPDAs property to True.
2. Code the OnPdaValidateRegToken and set Var TokenValid:Boolean to True.
3. Code the OnPdaValidatedPassword and set the Var PasswordValid:Boolean to True.

Authentication
Remote clients can be authenticated by the TAstaPdaServerSocket. To require password authentication set the AllowAnonymousPDAs property to False and code the OnPDAValidatePassword event. Remote PDA clients also generate a unique token, based on their hardware that can allow for authentication at the device level.

Encryption
ASTA PDA Clients can use either AstaAES or AstaDES Encryption.

Device Authentication
AstaPDA clients have the ability to generate a unique identier based on the hardware. This allows for SkyWire servers to be able to enforce licensing by only those devices registered on the server.

TAstaPdaServerSocket

Unit AstaPdaServersocket

Procedure PdaSendParamList(MsgID:Integer; ClientSocket:TCustomWinSocket; ParamList: TAstaParamList);

Description
PdaSendParamList sends an AstaParamList to remote a PDA Client. The AstaServerSocket call of SendcodedParamList will internally route any ParamLists to PDA clients and use PdaSendParamList so that SendCodedParamList can safely be used on any ASTA SkyWire server to communicate with any remote client.

The Msgid is used on the PDA Client when receiving the ParamList.

TAstaPdaServerSocket

Unit AstaPdaServersocket

property AllowAnonymousPDAs: **Boolean** default false;

Description
By default, authentication is required by the TAstaPdaServerSocket. To allow for remote PDA's to connect and communicate with an ASTA Server anonymously set the AllowAnonymousPDA property to true. There are also some events that need to be coded to allow for remote PDA clients to connect with no authentication.

TAstaPdaServerSocket

Unit AstaPdaServersocket

Procedure OnPalmUpateRequest(Sender: TObject; ClientSocket: TCustomWinSocket; P: TAstaParamList; **var** UpdatedVersion: **string**; **var** ErrorCode: **integer**) **of object**;

Description

TAstaPdaServerSocket

Unit AstaPdaServersocket

TAstaAcquireRegTokenEvent=**procedure**(Sender: TObject; U: TUserRecord; PDAid: **integer**; IntToken: TMessageDigest128; **var** UserToken: **string**) **of object**;

TAstaPdaServerSocket

Unit AstaPdaServersocket

Procedure OnPdaAuthentication(Sender: TObject; U: TUserRecord; Error: **integer**) **of object**;

TAstaPdaServerSocket

Unit AstaPdaServersocket

procedure OnPdaGetFileRequest(Sender: TObject; PDAid: **integer**; RequestString: **string**; StartPos, BlockSize: **integer**; Buffer: Pointer; **var** Written: **integer**; **var** ErrCode: **integer**) **of object**;

Description
Skywire servers can serve up file request to remote clients. Although the PalmOS does not have the concept of a file, a requested file can be stored in a Palm Database. This call allows for larger files to be read from servers in configurable "chunks" to get around any memory limitations on clients such as the 32K segment limitation on Palm's. WinCE clients have files and can request and send files using Asta Connection methods.

TAstaPdaServerSocket

Unit AstaPdaServersocket

Procedure OnPDAParamList(Sender:TObject; TheClient:TUserRecord;
Msgid:Integer; Params:TAstaParamList);

Description
The main processing engine for PDA requests which mapps to the AstaConnection.AstaConnSendParamList from remote PDA's.

TAstaPdaServerSocket

Unit AstaPdaServersocket

TAstaPDAPasswordNeededEvent=**procedure**(Sender: TObject; U: TUserRecord; UserName: **string**; **var** Password: **string**; **var** AccountValid: boolean) **of object**;

TAstaPdaServerSocket

Unit AstaPdaServersocket

TAstaRevokeRegTokenEvent=**procedure**(Sender: TObject; U: TUserRecord; PDAid: **integer**; IntToken: TMessageDigest128; **var** Result: boolean) **of object**;

TAstaPdaServerSocket

Unit AstaPdaServersocket

Procedure OnPdaValidatePassword(Sender:TObject; TheClient:TUserRecord; UserName, Password :String
Var PasswordValid:Boolean);

Description
Remote PDA clients, with Authentication set on, will send their UserName and Passwords which alllows authentication by setting the VAR PasswordValid to true or false.

TAstaPdaServerSocket

Unit AstaPdaServersocket

Procedure OnPdaValidateRegToken(Sender:TObject; InternalToken,ClientToken:String; var TokenValid:Boolean);

Description
Receives a token from a remote device that is guaranteed to be unique and represents a specific pda hardware client.

1.10.2.4.2.2 SkyWire Server API

Below are the standard events supported by ASTA SkyWire Servers with the ParamList filled with values that the server is expecting according to the conventions defined below. The tokens are defined in Constants.bas

These calls are initiated on the PDA clients by creating an AstaParamList and optionally adding values to it. and then ent to the server using a pre-defined token.

AstaConnSendParamList(RefLib, Conversation, ParamList, PdaServerTime, BoolRes);

PdaServerTime (1001)
This will retrieve the server time and can be used to see if the server is active like a ping

PdaSimpleSQLSelect (1003);
AstaPlSetString(RefLib, Result, 0, SQLString, 'PalmSelect');

One String Param is passed to the server and it is executed as Select SQL with one row being returned with each Field as a ParamName and the Value of each field contained AsString. To use Native Data use the full featured DataList token PdaMultiSelect.

PdaPalmTables (1004)

AstaPlSetString(RefLib, Result, 0, 'PalmTables', 'PalmTables');

Retrieves a list of tables from the server database. Servers could be coded to support more than one database in which case the String sent to the server could specify a particular database.

PdaPalmExecSQL (1005)

```
AstaPlSetString(RefLib, Result, 0, 'Exec1',UpdateSQLString);
AstaPlSetString(RefLib, Result, 1, 'Exec2',UpdateSQLString);
```

On the server, a transaction is started and each ParamItem is passed to an ExecSQL call. If all the ExecSQL's are successful, the transaction is committed, otherwise Rollback is called.

PdaPalmFieldDefs (1006)

```
AstaPlSetString(RefLib, Result, 0, 'Table','Customers');
```

The server expects one string param that contains the name of the table. A list of fields will be returned in an AstaParamList.

PdaNestedSQLSelect (1010)
Executes Select SQL at the server and returns an AstaNestedParamList

PdaMultiSQLSelect (1011)
Executes Select SQL at the server and returns an AstaDataList

PdaMultiExecSQLParamList (1012)

This allows for multiple parameterized queries handled by the server in a transaction so that blob and memo fields can be supported. Requires an AstaNestedparamList build since a ParamList must be sent along with each SQL Statement. Not currently implemented.

PdashowDigitalInk (1021)

This sends the graphical contents of a form to a server that is dislayed as a signature.

PdaGetFileFromServer (1050)

```
AstaPlSetString(RefLib, ParamList, 0, 'C:\readme.txt', 'FileName');
```

Gets a file from a remote server where the string param is the remote file name.

PdaSendFileToServer (1051)

```
AstaPlSetString(RefLib,ParamList, 0, 'C:\myfile.txt', 'FileName');
AstaPlSetBlob(RefLib,ParamList,1,TheBlob,BlobSize,'FileStream');
```

This sends a blob to a remote server and contains the FileName in Param Zero and the Stream to be saved in Param One as a Blob

SkyWireServerAPI

PdaNestedSQLSelect as defined in Constants.bas (1010) and part of the SkyWire Server API, ,

allows for AstaNestedParamLists to be requested from remote SkyWire servers. Use the CAstaRecordset.DataSetToParamList method to transfer a CAstaRecordSet to an AstaNestedParamList.

From the PDA 2 Params are required, the SQL Select statement and the RowCount. There are other optional params to control the

```
AstaPlSetString(RefLib, Result, 0, SQLString, 'PalmSelect');

AstaPlSetLongInt(RefLib, Result, 1, 5, 'MaxRows');
```

The first param will be passed as SQL on the server. The Second param, if defined, will be the maximum rows returned from the server. The server will return a result set as an AstaNestedParamList.

Constants.bas containst the constants used in pre-compiled ASTA SkyWire servers to support remote client side SQL Requests.

```
Public Const PDAErrorNone = 0
Public Const PDAGeneralError = 5000
Public Const PDAErrorSQL = PDAGeneralError + 1
Public Const PDAMetaDataError = PDAGeneralError + 2
Public Const PDAServerTime = 1001
Public Const PDASimpleSQLSelect = 1003
Public Const PDAPalmTables = 1004
Public Const PDAPalmExecSQL = 1005
Public Const PDAPalmFieldDefs = 1006
Public Const PDAStoredProcNames = 1007
Public Const PDAStoredProcParams = 1008
Public Const PDAExecuteStoredProc = 1009
Public Const PDANestedSQLSelect = 1010
Public Const PDAMultiSQLSelect = 1011
Public Const PDAMultiExecSQLParamList = 1012
Public Const PDAStoredProcDataList = 1013
Public Const PDAServerMethodList = 1014
Public Const PDAServerMethodParams = 1015
Public Const PDASevermethodExec = 1016
Public Const PDASeverMethodDataList = 1017
Public Const PDAProviderList = 1018
Public Const PDAProviderParams = 1019
Public Const PDAProviderDataList = 1020
Public Const PDAshowDigitalInk = 1021
Public Const PDAGetFileFromServer = 1050
Public Const PDASendFileToServer = 1051
```

SkyWireServerAPI

SkyWire Servers are available that support the SkyWire API which includes standard tokens as defined in
Constants.bas that allow remote PDA clients to use client side SQL.

These calls are initiated on the PDA clients by creating an AstaParamList and optionally adding values to it. It is then sent to the server using the PdaServerTime Token

```
AstaConnSendParamList(RefLib, Conversation, ParamList, PdaServerTime, BoolRes);
```

This will retrieve the server time and can be used to see if the server is active like a ping

SkyWire clients can be run on many platforms including Palm, Wince and Linux Pda's as well as Win32 via COM of with VC++ and any Linux using Kylix or a C++ class using the gcc Linux compiler. Below you will find a couple of code examples.

NSBASIC for the Palm

CASL for the Palm

Codewarrior for the Palm

PocketStudio for the Palm

eVB for WinCE

eVC++ for WinCE

C# for DOT NET

Satellite Forms for the Palm

```
CAstaClientConnection conn = new CAstaClientConnection();
conn.ServerVersion = settings.isAstaIO ? CAstaClientConnection.ServerVersionAstaIO :
CAstaClientConnection.ServerVersionAsta;
                                byte[] token = new byte[32];
                                for (int i = 0; i < 32; i++)
                                        token[i] = 0;
                                conn.SetRegToken(token);

                                b = conn.OpenConnection(settings.ServerAddress,
settings.ServerPort);
                                if (b)
                                {
                                        b = conn.SendParamList((int) Request, Params);
                                        if (b)
                                        {
                                                Params = conn.ReceiveParamList();
                                                b = Params != null;
                                        }
                                        if (b)
                                        {
                                                if (Request != AstaSQLToken.Select && Request !=
AstaSQLToken.MultiSelect)
                                                {
                                                        CAstaParamItem param;
                                                        string s1, s2;
                                                        ResultList.Items.Clear();
                                                        for (int i = 0; i < Params.Count; i++)
                                                        {
                                                                param = (CAstaParamItem)
Params[i];
                                                                if (param != null)
                                                                {
                                                                        s2 = param.AsString;
                                                                        if (s2 == null)
                                                                                s2 = "non-string
parameter";
                                                                        s1 = param.Name + " = " +
s2;
                                                                        ResultList.Items.Add(s1);
                                                                }
                                                        }
                                                }
                                                else
                                                {
                                                }
                                        }
                                        conn.CloseConnection();
                                }
                        }
                }
```

This is an example of sending an AstaParamList to an AstaSkyWire server using CASL.

```
Function SendAPL(numeric MsgID, String SQLSelect);
  # Reset the currently selected table (used by the [Fields] button)
  Table = "";

  # Retrieve user selections For addressing
  Get ServerAddressText, Address;
  Get ServerPortText, Port;

  ClearSelector;
  hParams = AstaPLCreate;

  # As well As creating an Asta Param List, we must add an initial item To avoid
  # nil pointer errors at runtime
  AstaPLSetString(hParams, 0, SQLSelect,"SQL Select" );


  hConnection = AstaClCnvCreate;

  If AstaClCnvOpenConnection(hConnection, Address, Port) <> 0;
    If AstaClCnvSendParameters(hConnection, hParams, MsgID) <> 0;
          AstaPLClear(hParams);

          If AstaClCnvReceiveParameters(hConnection, hParams) <> 0;
            If AstaPLGetCount(hParams) > 16;
                  APLCount = 16;
              Else;
                APLCount = AstaPLGetCount(hParams);
              End_If;

        sDataFromServer.selected=0;
        sDataFromServer.list_size = APLCount;

        For i = 0, i < APLCount;
                 apllist[i] = AstaPLGetAsString(hParams, i);
              Next i + 1;
          Else;
            Put Label16, "Receive Params Fail";
          End_If;
        Else;
       Put Label16, "Send Params Fail";
        End_If;
  Else;
    Put Label16, "Asta Connect Fail";
  End_If;

  Hide sDataFromServer;
  Show sDataFromServer;

  AstaClCnvDestroy(hConnection);
  AstaPLDestroy(hParams);
End;
```

```
err = AstaConnCreate(AstaLibRef, &conn);

                err = AstaConnSetTimeout(AstaLibRef, conn, 10000);

                err = AstaConnOpen(AstaLibRef, conn, szAddress, nPort, &bResult);

                if (bResult)
                {
                        // enabling/disabling authorization
                        if (bUseAuthorization)
                                AstaConnSetAuthorizationOn(AstaLibRef, conn, szUserName,
szPassword, true);
                        else
                                AstaConnSetAuthorizationOff(AstaLibRef, conn);

                        // sending the request
                        err = AstaConnSendParamList(AstaLibRef, conn, params, Request,
&bResult);

                        if (!err && bResult)
                        {
                                // receiving a reply
                                AstaPLClear(AstaLibRef, params);

                                AstaConnReceiveParamList(AstaLibRef, conn, params, &bResult);

                                if (bResult)
                                {

                                        if (Request != PalmSelect && Request != PalmMultiSelect)
                                        {
                                                FormPtr frmP = FrmGetActiveForm();
                                                // clearing the list
                                                ListPtr lstP = (ListPtr)FrmGetObjectPtr(frmP,
FrmGetObjectIndex(frmP, SQLResultsList));
                                                LstSetListChoices(lstP, NULL, 0);

                                                // displaying the data received

                                                long count;
                                                AstaPLGetCount(AstaLibRef, params, &count);

                                                CharPtr *list = (char**)MemPtrNew(count *
sizeof(CharPtr));


                                                if (count > 0)
                                                {
                                                        for (int i = 0; i < count; i++)
                                                        {
                                                                VoidPtr param;
                                                                AstaPLGetParameter(AstaLibRef,
params, i, &param);
                                                                if (!param)
                                                                        list[i] = NULL;
                                                                else {
                                                                        long NameLen = 0, ValueLen
= 0;

                                                                        CharPtr lpName = NULL;
                                                                        AstaPLParamGetName(AstaLibR
ef, param, lpName, &NameLen);

                                                                        lpName = (CharPtr)
MemPtrNew(NameLen + 1);

                                                                        AstaPLParamGetName(AstaLibR
ef, param, lpName, &NameLen);


                                                                        CharPtr lpValue = NULL;
                                                                        AstaPLParamGetAsString(Asta
LibRef, param, lpValue, &ValueLen);

                                                                        lpValue = (CharPtr)
MemPtrNew(ValueLen + 1);

                                                                        AstaPLParamGetAsString(Asta
LibRef, param, lpValue,
&ValueLen);
```

```
                                                                CharPtr newParam;

                                                                if (lpValue) {
                                                                        newParam =
(CharPtr)MemPtrNew(StrLen(lpName) + 3 + StrLen(lpValue) + 1);
                                                                        StrCopy(newParam,
lpName);
                                                                        StrCat(newParam, "
= ");
                                                                        StrCat(newParam,
lpValue);
                                                                        MemPtrFree(lpValue)
;
                                                                }
                                                                else {
                                                                        newParam =
(CharPtr)MemPtrNew(StrLen(lpName) + 24);
                                                                        StrCopy(newParam,
lpName);
                                                                        StrCat(newParam, "
= non-string parameter");
                                                                }
                                                                if (lpName)
                                                                        MemPtrFree(lpName);


                                                                list[i] = newParam;
                                                        }
                                                }
                                                LstSetListChoices(lstP, list, count);
                                        }

                                        LstDrawList(lstP);
                                        AstaPLDestroy(AstaLibRef, params);
                                }
                                else
                                if (Request == PalmMultiSelect)
                                {

                                        ListPtr lstP = (ListPtr)FrmGetObjectPtr(frmP,
FrmGetObjectIndex(frmP, SQLResultsList));
                                        LstSetListChoices(lstP, NULL, 0);

                                        VoidPtr Data;
                                        long DataLen;
                                        CharPtr StrValue, lpItem, lpReal;
                                        VoidPtr Value;


                                        AstaPLGetParameter(AstaLibRef, params, 0,
&Value);


                                        Data = NULL;
                                        DataLen = 0;
                                        AstaPLParamGetAsBlob(AstaLibRef, Value, Data,
&DataLen);

                                        Data = MemPtrNew(DataLen);
                                        AstaPLParamGetAsBlob(AstaLibRef, Value, Data,
&DataLen);


                                        AstaPLDestroy(AstaLibRef, params);

                                        VoidPtr List;
                                        AstaDLCreate(AstaLibRef, Data, DataLen, &List);

                                        MemPtrFree(Data);

                                        long ccount, count, StrValueLen;
                                        AstaDLGetRecordsCount(AstaLibRef, List, (unsigned
long *) &ccount);


                                        CharPtr *list = (char**)MemPtrNew((ccount + 1) *
sizeof(CharPtr));
```

```
                                                    MemSet(list, (ccount + 1) * sizeof(CharPtr), 0);

                                                    for (UInt32 i = 0; i < ccount; i++)
                                                    {
                                                            lpItem = (CharPtr) MemPtrNew(1400);
                                                            *lpItem = 0;

                                                            AstaDLGetAsParamList(AstaLibRef, List, i,
true, &params);

                                                            if (params)
                                                            {
                                                                    AstaPLGetCount(AstaLibRef, params,
&count);

                                                                    if (count > 0)
                                                                    {
                                                                            AstaPLGetParameter(AstaLibR
ef, params, 0, &Value);

                                                                            StrValueLen = 0;
                                                                            AstaPLParamGetAsString(Asta
LibRef, Value, NULL, &StrValueLen);

                                                                            StrValue = (CharPtr)
MemPtrNew(StrValueLen + 1);

                                                                            AstaPLParamGetAsString(Asta
LibRef, Value, StrValue, &StrValueLen);



                                                                            if (StrValue)
                                                                            {
                                                                                    StrCopy(lpItem,
StrValue);

                                                                                    MemPtrFree(StrValue
);
                                                                            }



                                                                            long k;

                                                                            AstaDLGetFieldCount(AstaLib
Ref, List, &k);

                                                                            for (long j = 1; j < k;
j++)
                                                                            {
                                                                                    AstaPLGetParameter(
AstaLibRef, params, j, &Value);

                                                                                    StrValueLen = 0;
                                                                                    AstaPLParamGetAsStr
ing(AstaLibRef, Value, NULL, &StrValueLen);

                                                                                    StrValue =
(CharPtr) MemPtrNew(StrValueLen + 1);

                                                                                    AstaPLParamGetAsStr
ing(AstaLibRef, Value, StrValue, &StrValueLen);


                                                                                    if (StrValue)
                                                                                    {
                                                                                            StrCat(lpIte
m, " ");

                                                                                            StrCat(lpIte
m, StrValue);

                                                                                            MemPtrFree(S
trValue);
                                                                                    }
                                                                            }
                                                                    }
                                                                    else
                                                                            *lpItem = 0;
                                                                    if (params)
                                                                            AstaPLDestroy(AstaLibRef,
params);
                                                    }
                                                    lpReal = (CharPtr)
MemPtrNew(StrLen(lpItem) + 1);
```

```
                                                StrCopy(lpReal, lpItem);
                                                MemPtrFree(lpItem);
                                                list[i] = lpReal;
                                        }
                                        LstSetListChoices(lstP, list, ccount);
                                        LstDrawList(lstP);
                                        AstaDLDestroy(AstaLibRef, List);
                                }
                                else
                                if (Request = PalmSelect)
                                {
                                        ListPtr lstP = (ListPtr)FrmGetObjectPtr(frmP,
FrmGetObjectIndex(frmP, SQLResultsList));
                                        LstSetListChoices(lstP, NULL, 0);



                                        AstaNPLCreate(AstaLibRef, params, &cData);
                                        showData();
                                        AstaPLDestroy(AstaLibRef,
params);

                                }
                        }
                        else
                        {
                                long err;
                                AstaConnGetLastError(AstaLibRef, conn, &err);
                                if (err >= AuthenticationError && err <=
NoAuthenticationError)
                                  FrmAlert(AccessDeniedAlert);
                        }
                }
                bool result;
                AstaConnClose(AstaLibRef, conn, &result);
                AstaConnDestroy(AstaLibRef, conn);

        }
    }

    return bResult;
}
```

Below is some sample code showing how to return an AstaDatalist from an SQL select using embedded Visual BASIC for WinCE.

```
Private Sub CmdSubmit_Click()
    Dim ParamList As CoAstaParamList
    Dim ResultArray As CoAstaDataArray
    Dim ResultList As CoAstaParamList
    Dim ResEnum As CoEnumAstaParam
    Dim ParamItem As CoAstaParamItem
    Dim Field As CoAstaField
    Dim LogLine As String
    Dim I As Integer
    Dim ResultMatrix As CoAstaMatrixParamList
    Dim FieldItem As CoAstaParamItem

    'Send request to server
    Set ParamList = FrmConnect.Connection.CreateParamList
    Set ParamItem = ParamList.AddItem
    ParamItem.AsString = TxSQL.Text
    ParamItem.Name = "WinCESelect"
    Set ParamItem = ParamList.AddItem
    ParamItem.AsInteger = 1000

    If ChDataList.Value Then
        FrmConnect.Connection.SendParamList ParamList, 1011
        Set ParamItem = Nothing
        Set ParamList = Nothing

        'Receive data array
        Set ResultArray = FrmConnect.Connection.ReceiveDataArray

        'Show received header
        LogLine = ""

        For I = 0 To ResultArray.FieldCount - 1
            Set Field = ResultArray.GetFieldByIndex(I)
            LogLine = LogLine + Field.Name + "   "
        Next
        logMessage LogLine
        Set Field = Nothing 'cleanup

        'Show received data
        For I = 0 To ResultArray.RecordCount - 1
            Set ResultList = ResultArray.GetRecord(I)
            Set ResEnum = ResultList.GetEnum
            LogLine = ""
            While ResEnum.HasMore
                Set ParamItem = ResEnum.Next
                LogLine = LogLine + "   " + ParamItem.AsString
            Wend
            logMessage LogLine
            Set ParamItem = Nothing
            Set ResEnum = Nothing
            Set ResultList = Nothing
        Next

        Set ResultArray = Nothing
    Else
        FrmConnect.Connection.SendParamList ParamList, 1010
        Set ParamItem = Nothing
        Set ParamList = Nothing

        Set ResultMatrix = FrmConnect.Connection.ReceiveMatrixParamList
        While Not ResultMatrix.isEOF
            LogLine = ""
            For I = 0 To ResultMatrix.FieldCount - 1
                Set FieldItem = ResultMatrix.FieldItem(I)
                LogLine = LogLine + FieldItem.Name + "|" + FieldItem.AsString + "   "
            Next
            logMessage LogLine
            ResultMatrix.Next
```

```
Wend

        While Not ResultMatrix.isBOF
            LogLine = ""
            For I = 0 To ResultMatrix.FieldCount - 1
                Set FieldItem = ResultMatrix.FieldItem(I)
                LogLine = LogLine + FieldItem.Name + "|" + FieldItem.AsString + "   "
            Next
            logMessage LogLine
            ResultMatrix.Prev
        Wend
        Set FieldItem = Nothing
        Set ResultMatrix = Nothing
    End If

End Sub
```

This is from a SkyWireTest Client demo that shows how to create an AstaParamList, send it to the server, request the server time and show the results on a WinCE device.

```
if ((HWND) lParam == hwndServerTimeButton)
                {
                        ParamLst = AstaParamLstCreate();
                        AstaParamLstSetString(ParamLst, 0, TEXT("NA"), TEXT("ServerTime") );
                        if (!ConnPresent)
                                DoConnect();
                        if (ConnPresent)
                                if (!AstaClConvSendParamList(ClientConn, 1001, ParamLst))
                                {
                                        MessageBox(hwndMain, TEXT("Failed to send request"),
NULL, 0);
                                        ConnPresent = false;
                                        SetWindowText(hwndStatusLabel, TEXT("Disconnected"));
                                }
                        if (ConnPresent)
                        {
                                AstaParamLstClearParameters(ParamLst);
                                if (!AstaClConvReceiveParamList(ClientConn, ParamLst))
                                {
                                        MessageBox(hwndMain, TEXT("Failed to receive response"),
NULL, 0);
                                        ConnPresent = false;
                                        SetWindowText(hwndStatusLabel, TEXT("Disconnected"));
                                }

                        }
                        if (ConnPresent)
                        {
                                ShowResults1(ParamLst, hwndServerTimeList);
                        }
                        AstaParamLstDestroy(ParamLst);
                        return TRUE;
                }
```

NSBASIC

This code just checks to see if a server is available. For more detailed information see the AstaSkyWireNSBASIC.hlp file.

this code must be included in the Startup section of the NSBASIC project

```
Sub main()
  LoadLibrary "AstaLib"
End Sub



Sub object1013()
  Dim Conn As Integer
  Dim R As Byte
  Dim R1 As Byte
  Dim Result As Integer
  Dim S As String

  Conn = AstaLib.AstaConnCreate()
  Address = AddressField.Text
  Port = Val(PortField.Text)
  R = AstaLib.AstaConnOpen(Conn, Address, Port)

  If R = Chr(0) Then
    Result = Alert("AstaSQL", "IP Address or Port number is invalid", 0, "OK")
  Else
    R1 = AstaLib.AstaConnClose(Conn)
    If UseAuthorizationCheckbox.Status = nsbChecked Then
      UseAuthorization = 1
    Else
      UseAuthorization = 0
    End If
    UserName = UserNameField.Text
    Password = PasswordField.Text
  End If

  AstaLib.AstaConnDestroy(Conn)

  If R <> Chr(0) Then
    NextScreen "QueryForm"
  End If

End Sub
```

This is taken from a PocketStudio demo that shows how to use the AstaDataList. An AstaConnection component is created, a connection is made to the server, an AstaParamList is created and filled with data, sent to the server with error checking done and the results displayed as an exception of there is an error, otherwise processed normally.

```
procedure AstaConversation(ConvType:UInt16);
var
  Conversation, ParamList: Pointer;
  ErrorNum:Int32;
  BoolRes : boolean;
  ErrorMsg:Pointer;
begin
  AstaConnCreate(RefLib, Conversation);
  //AstaConnSetAuthorizationOn(RefLib,Conversation,'ASTA','Password',False);
  AstaConnOpen(RefLib, Conversation, Connection.IPAddr, Connection.Port, BoolRes);
  if BoolRes then
  begin
    ParamList := PrepareParameters(ConvType);

    AstaConnSendParamList(RefLib, Conversation, ParamList, ConvType, BoolRes);
    if not BoolRes then
```

```
begin
      FrmCustomAlert(UAlert, 'Failed to send the data', '', '')
    end
    else
    begin
      AstaPlClear(RefLib, ParamList);
      ErrorNum:=0;
      AstaConnReceiveParamList(RefLib, Conversation, ParamList, BoolRes);

      AstaConnGetLastError(RefLib,Conversation,ErrorNum);
      AstaConnGetUserError(RefLib,Conversation,ErrorNum);
      if ErrorNum<>0 then begin
       BoolRes:=False;
       AstaShowException(reflib,ParamList);
      end;
      if (not BoolRes) then begin
        if ErrorNum=0 then FrmCustomAlert(UAlert, 'Failed to receive the reply', '', '')
      end
      else
      begin
        if ConvType <> PdaMultiSQLSelect then
          UpdateReplyList(ParamList)
        else
          UpdateSelectList(ParamList);
      end;
    end;

    AstaConnClose(RefLib, Conversation, BoolRes);
    AstaPlDestroy(RefLib, ParamList);
  end
  else
  begin
    FrmCustomAlert(UAlert, 'Failed to connect to server', '', '')
  end;

  AstaConnDestroy(RefLib, Conversation);
end;
```

```
Sub DoConversation(Request)
	Dim bResult
	Dim Params
	Dim err
	Dim StrValue
	Dim ConnHandle
	Dim RowCount
	Dim CurRow
	Dim FieldCount
	Dim CurField
	Dim CurValue
	Dim CurParam
	Dim LHandle
	Dim Buffer
	Dim BufSize
	Params = Extensions("AstaLib").AstaPLCreate(2)
	If Request = 1001 Then
		Extensions("AstaLib").AstaPLSetString(Params, 0, "NA", "ServerTime")
	ElseIf Request = 1004 Then
		Extensions("AstaLib").AstaPLSetString(Params, 0, "NA", "PalmTables")
	ElseIf Request = 1006 Then
		StrValue = Forms("MainForm").Controls("TableEdit").Data
		Extensions("AstaLib").AstaPLSetString(Params, 0, StrValue, "PalmFields")
	ElseIf (Request = 1010) Or (Request = 1003) Then
		StrValue = Forms("MainForm").Controls("SelectEdit").Data
		Extensions("AstaLib").AstaPLSetString(Params, 0, StrValue, "PalmSelect")
		Extensions("AstaLib").AstaPLSetLongInt(Params, 1, 9999, "RowCount")
	endif
	ConnHandle = Extensions("AstaLib").AstaConnCreate
	bResult = Extensions("AstaLib").AstaConnOpen(ConnHandle, ServerAddress, ServerPort)
	If bResult Then
	    If Authorize <> 0 Then
			Extensions("AstaLib").AstaConnSetAuthorizationOn(ConnHandle, Username,
Password, True)
		Else
			Extensions("AstaLib").AstaConnSetAuthorizationOff(ConnHandle)
		endif
		bResult = Extensions("AstaLib").AstaConnSendParamList(ConnHandle, Params,
Request)
		If bResult <> 0 Then
			Extensions("AstaLib").AstaPlClear(Params)
			bResult = Extensions("AstaLib").AstaConnReceiveParamList(ConnHandle,
Params)
			If bResult <> 0 Then
				Forms("MainForm").Controls("ResultEdit").Data = ""
				If (Request <> 1010) And (Request <> 1003) Then
					RowCount = Extensions("AstaLib").AstaPLGetCount(Params)
					CurValue = ""
					For CurRow = 0 To RowCount - 1
						CurParam = AstaPlGetParameter(Params, CurRow)
						StrValue =
Extensions("AstaLib").AstaPlParamGetName(CurParam)
						CurValue = StrValue
						StrValue =
Extensions("AstaLib").AstaPlParamGetAsString(CurParam)
						CurValue = CurValue & " = " & StrValue
						If (Forms("MainForm").Controls("ResultEdit").Data
= "") Then
							Forms("MainForm").Controls("ResultEdit").D
ata = CurValue
						Else
							Forms("MainForm").Controls("ResultEdit").D
ata = Forms("MainForm").Controls("ResultEdit").Data & Chr(10) & CurValue
						endif
					Next CurRow

				ElseIf Request = 1010 Then
					'NestedParamList
					LHandle = Extensions("AstaLib").AstaNPLCreate(Params)
					RowCount =
Extensions("AstaLib").AstaNPLGetRecordsCount(LHandle)
					If RowCount > 0 Then
```

```
                                        FieldCount =
Extensions("AstaLib").AstaNPLGetFieldCount(LHandle)
                                        CurValue = ""
                                        For CurField = 0 To FieldCount -1
                                                CurParam =
Extensions("AstaLib").AstaNPLGetField(LHandle, CurField)
                                                If (CurValue = "") Then
                                                        CurValue =
Extensions("AstaLib").AstaPLParamGetName(CurParam) & " = " &
Extensions("AstaLib").AstaPlParamGetAsString(CurParam)
                                                Else
                                                        CurValue = CurValue & Chr(10) &
Extensions("AstaLib").AstaPLParamGetName(CurParam) & " = " &
Extensions("AstaLib").AstaPlParamGetAsString(CurParam)
                                                endif
                                        Next CurField
                                        Forms("MainForm").Controls("ResultEdit").Data =
CurValue
                                endif
                                AstaNPLDestroy(LHandle)
                        Else
                                'DataList
                                RowCount = Extensions("AstaLib").AstaPLGetCount(Params)
                                If RowCount > 0 Then
                                        CurParam =
Extensions("AstaLib").AstaPLGetParameter(Params, 0)
                                        Buffer =
Extensions("AstaLib").AstaPlParamGetAsBlob(CurParam)
                                        BufSize =
Extensions("AstaLib").AstaGetBufferSize(Buffer)
                                        LHandle = AstaDLCreate(Buffer, BufSize)
                                        Extensions("AstaLib").AstaReleaseBuffer(Buffer)
                                        RowCount =
Extensions("AstaLib").AstaDLGetRecordsCount(LHandle)
                                        If RowCount > 0 Then
                                                CurValue = ""
                                                For CurRow = 0 To RowCount -1
                                                        AstaPLDestroy(Params)
                                                        Params =
AstaDLGetAsParamList(LHandle, CurRow, 1)
                                                        FieldCount =
Extensions("AstaLib").AstaDLGetFieldCount(LHandle)
                                                        For CurField = 0 To FieldCount - 1
                                                                CurParam =
Extensions("AstaLib").AstaPlGetParameter(Params, CurField)
                                                                CurValue = CurValue &
Extensions("AstaLib").AstaPlParamGetName(CurParam) & " = " &
Extensions("AstaLib").AstaPlParamGetAsString(CurParam)
                                                        Next
CurField
                                                        If
(Forms("MainForm").Controls("ResultEdit").Data <> "") Then
                                                                CurValue =
Forms("MainForm").Controls("ResultEdit").Data & Chr(10) & CurValue
                                                        endif
                                                Next CurRow
                                                Forms("MainForm").Controls("ResultEdit").D
ata = CurValue
                                        endif
                                        AstaDLDestroy(LHandle)
                                endif

                        endif
                Else
                        MsgBox("Failed to receive reply")
                endif
                AstaPLDestroy(Params)
        Else
                MsgBox("Failed to send request")
        endif
        bResult = Extensions("AstaLib").AstaConnClose(ConnHandle)
Else
        MsgBox("Failed to connect to server")
endif
Extensions("AstaLib").AstaConnDestroy(ConnHandle)
```

```
End Function
```

SkyWireServerAPI

PdaMultiSQLSelect as defined in Constants.bas (1011) and part of the SkyWire Server API, , allows for AstaDatalists to be requested from remote SkyWire servers. Use the CAstaRecord.DataSetToDataList method to transfer a CAstaRecordSet to an AstaDataList.

From the PDA 2 Params are required, the SQL Select statement and the RowCount. There are other optional params to control the

AstaPlSetString(RefLib, Result, 0, SQLString, 'PalmSelect');

AstaPlSetLongInt(RefLib, Result, 1, 5, 'RowCount');

The first param will be passed as SQL on the server. The Second param, if defined, will be the maximum rows returned from the server. The server will return a result set as an AstaDataList. There are additinal Params that can be defined to further customize the amount of data coming from the server. These params must be named according to the the following conventions. The first param must contain the SQL. The second param can optionally contain the rowcount. The rest of the options can be sent in any order and all of the options need not be sent but the ParamNames must correspond exactly to those below.

EndRow AstaPlSetLongInt(RefLib, Result, 2, 5, 'EndRow');

This sets the row to End at or the Rowcount. If not defined all rows will be returned which is equivalent to passing in -1.

StartRow AstaPlSetLongInt(RefLib, Result, 2, 1, 'StartRow');

This sets the row to start at to packup the result set. the default is row 1.

MaxStringLength AstaPlSetLongInt(RefLib, Result, 3, 100, 'MaxStringLength');

The maximum length of any string field. By default the complete string will be included in the datalist.

MaxTotalLength AstaPlSetLongInt(RefLib, Result, 4, 15000, 'MaxTotalLength');

The maximum size of the DataList. Since the Palm cannot handle more than 32K of data in one datasegment you must not use a number larger than 32,767

1.10.2.4.2.3 RecordSets

In order to be able to manage SQL result sets, or "recordsets", Asta has developed 2 data structures that are available to receive SQL result sets from remote databases.These data structures can be created on SkyWire servers by the CAstaRecordSet that has calls to convert server side RecordSets to AstaNestedParamLists or to AstaDataLists.

AstaNestedParamList
AstaDataList

These data structures will eventually be integrated with with ASTA SkySync technology that will allow for Pda's to easily and efficiently resolve their changed data to remote databases.

The AstaNestedParam is an all string DataStructure that handles rows and columns much like a database table. On pre-compiled servers, or servers that support the SkyWire Server API, the token PDANestedSQLSelect (1010) is mapped to the server side method.

The AstaNestedParamList provides a client side datatructure that works like a RecordSet with a row/column abstraction in order to handle SQL result requests from ASTA SkyWire servers with all the data stored as Strings. This makes it easy to use with User Interface Controls. To handle data natively, and more efficiently use AstaDataLists. NestedParamLists are actually an AstaParamList of AstaParamLists with the first ParamList a list of Fields and the rest of the paramlists containing the actual SQL Result sets.

AstaNestedParamLists can be created on the server by calling the CAstaRecordSet RecordToParamList method.

The AstaDataList is an efficient DataStructure that handles rows and columns much like a database table. On pre-compiled servers, or servers that support the SkyWire Server API, the token PdaMultiSQLSelect (1011) is mapped to the server side method.

The DataList basically contains a List of ParamLists that contain data natively, stored very efficiently, wthout duplicating paramnames, like the AstaNestedParamList does. Each row can be retrieved into a ParamList with the option of bring the Param or "FieldNames" also by calling AstaDLGetAsParamList

AstaDataLists can be created on the server by calling the CAstaRecordSet 'DataSetToDataList method.

See ASTA SkyWire client side documentation for more details on AstaDataLists.

1.10.2.4.2.4 Business Logic

You can certainly use the SkyWire Server API to allow for your remote clients to execute SQL but you also can add Business Logic to your own ASTA server by customizing your SkyWire Server. In this way, your clients always make the same call to the server but you can always change the implementation by changing the server and not worry about the clients having to be changed. Of course ASTA has an Auto Client Update feature across all platforms to make sure that all remote SkyWire clients are up to date.

Below are a couple of examples of Business Logic as coded on the example ASTA Visual Basic Northwind Server.

1.10.2.4.2.5 Error Handling

Remote ASTA PDA Clients make requests to Asta SkyWire servers by creating and filling up an AstaParamList with data and sending it over the wire along with a token that is used to key action on

the server. The server, routes the AstaParamList filled with data, based on the token and processes it. If that data can not be processed, like an SQL statement with a syntax error, an ErrorCode is returned to the Pda Client along with an error message as a string.

Here is some VB Code on how to create react to a problem on the server:

```
Private Sub AstaServer_OnParamList(ByVal MsgID As Long, ByVal InParamList As
AstaCOMServer.ICAstaParamList, OutParamList As AstaCOMServer.ICAstaParamList, ErrorCode As
Long, ErrorMessage As String)

    ErrorCode = PDAErrorSQL
   ' from Constants.bas
    ErrorCodeMessage = "SQL Syntax Error"

End_Sub
```

Here is an example of how a client deals with this error message using eVC++ for Wince

```
AstaParamLstClearParameters(Params);
AstaClConvReceiveParamList(ClientConn, Params);
if (AstaClConvGetUserError(ClientConn) != pdaErrorNone)
{
AstaHandle CurParam =  AstaParamLstGetParameter(Params, 0);
TCHAR ErrBuf[200];
DWORD ErrLen = 200;
AstaParamGetAsString(CurParam, ErrBuf, &ErrLen);
MessageBox(0, ErrBuf, NULL, 0);
}
```

1.10.2.4.2.6 Starting a Server

CAstaComServer

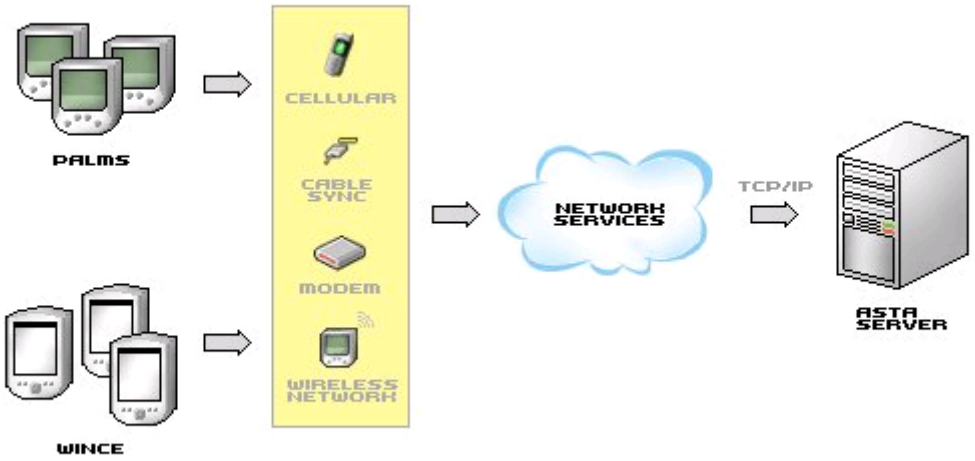
Below is some code from a SkyWire Server written with Visual Basic.

The Server is set to not authenticate remote clients and the port is set to 9090 and then the Active property is set to true. The server is now available to take requests from remote clients.

```
Private Sub Form_Load()

    Set AstaServer = New CAstaCOMServer
    Set UserDataSet = New CAstaRecordSet

    TbOpt.TabIndex = 0
    FrmMain.Caption = "ASTA SkyWire Server listening on " & AstaServer.GetThePCIPAddress & "
port " _
    & Str(AstaServer.Port)

    AstaServer.Port = 9090
    AstaServer.AllowAnonymousPDAs = True
    AstaServer.Active = True


    End With

End Sub
```

1.10.2.4.3 Clients

1.10.2.4.3.1 CASL



The ASTA SkyWire architecture consists of remote thin clients, connecting to a middle tier Application Server via TCP/IP. SkyWire Servers are available using a number of languages include Visual Basic, Delphi and Java. This document describes how to use CASL Clients to access remote SkyWire Servers servers and perform standard tasks using the SkyWire component.

ASTA Skywire consists of a middleware Server and client side components to access that server. The following components are used by SkyWire clients to perform standard tasks.

AstaConnection
AstaParamList
AstaParamValue
AstaDataList
AstaNestedParamList

The AstaClientConnection is the client end of the TCP/IP connection. It connects the ASTA client to the ASTA SkyWire Server. The Address and Port properties on the ASTA client must match the Address and Port properties on the ASTA SkyWire Server. Typically the handheld device would have a form where the IPAddress and Port can be input by the user so that the AstaClientConnection can connect to a SkyWire server and authenticate using a Username and Password.

AstaConnClose
AstaConnCreate
AstaConnDestroy
AstaConnGetFile
AstaConnGetLastError
AstaConnGetUserError
AstaConnOpen
AstaConnReceiveParamList
AstaConnSendParamList

AstaConnSetAuthenticationOff
AstaConnSetAuthenticationOn
AstaConnSetCompressionOff
AstaConnSetCompressionOn
AstaConnSetEncryptionOff
AstaConnSetEncryptionOn
AstaConnSetHttpOn
AstaConnSetHttpOff
AstaConnSetTimeout
AstaRequestUpdate

AstaConnection

function AstaConnClose(refNum: UInt16; hConnInstance: Pointer; **var** Result: boolean): Err;

Description

Closes a connection opened after calling AstaConnOpen.

Example

```
AstaConnClose(RefLib, Conversation, BoolRes);
```

AstaConnection

function AstaConnCreate(refNum: UInt16; **var** Result: Pointer): Err;

Description
Creates an AstaConnection as the Var Result:Pointer where refNum represents the handle to the Asta Shared Library.

Example
```
AstaConnCreate(RefLib, Conversation);
```

AstaConnection

function AstaConnDestroy(refNum: UInt16; hConnInstance: pointer): Err;

Description
Destroys and AstaConnection and releases it's resources that has been created with AstaConnOpen.

Example
```
AstaConnDestroy(RefLib, Conversation);
```

AstaConnection

function AstaConnGetFile(refNum :UInt16; hConnInstance :Pointer; BlockSize:UInt32;
RemoteFile:PChar; LocalName :PChar; Var Result:Boolean):Err

Description
Retrieves a remote file from an ASTA Server in BlockSize "chunks". This allows, for example, for large

images to be streamed down efficiently.

Example
AstaConnGetFile(refLib, Connection, 8192, RemoteFileName, LocalFileName, Result);

AstaConnection

function AstaConnGetLastError(refNum: UInt16; hConnInstance: Pointer; **var** Result: Int32): Err;

Description
Returns the last error generated by the AstaClientConnection. AstaSky Wire servers are codedso as to return an Integer value on each client request where a non-zero value is considered to be an error.

Example
AstaConnGetLastError(RefLib,Connection,Result)

AstaConnection

function AstaConnGetUserError(refNum: UInt16; hConnInstance: Pointer; **var** Result: Int32): Err;

escription
Returns the last error generated by the AstaClientConnection. AstaSky Wire servers are coded so as to return an Integer value on each client request where a non-zero value is considered to be an error. Typically errors are also returned with the server side error message or exception in an AstaParamValue of Name "Error"

Example
AstaConnGetUserError(RefLib,Connection,Result);

AstaConnection

function AstaConnOpen(refNum: UInt16; hConnInstance: Pointer; Address: PChar; Port: **Word**; **var** Result: boolean): Err;

Description
Attempts to connect to a remote AstaSkyWire server using the Address and Port and will return a False a connection to the server cannot be achieved. No data is transferred to the server. The Asta SkyWire shared library must already be loaded.

Example
```
AstaConnOpen( RefLib, Conversation, '127.0.01',9050, BoolRes);
```

AstaConnection

function AstaConnReceiveParamList(refNum: UInt16; hConnInstance: Pointer; hPLInstance: Pointer; **var** Result: boolean): Err;

Description
Receives an AstaParamList from a remote ASTA Server where the hPLIntance is an AstaParamList that has been created with a call to AstaPLCreate. If the ParamList was sucessfully retrieved from the

server the Var Result:Boolean would be set to True.

Example
Typically, if you had an existing Paramlist that you had used in sending params to a remote server you would want to clear it first.

```
AstaPlClear(RefLib, ParamList);
AstaConnReceiveParamList(RefLib, Conversation, ParamList, BoolRes);
```

AstaConnection

function AstaConnSendParamList(refNum: UInt16; hConnInstance: Pointer; hPLInstance: Pointer; MsgToken: UInt32; **var** Result: boolean): Err;

Description
Sends an AstaParam to a remote Server. The AstaParamList must have been created and typically is filled with values before being sent to the remote server. The MsgToken:UInt32 will determine how the server processes the request. If the ParamList was successfully sent to the server, teh Var Result:Boolean will be set to true.

Example
AstaConnSendParamList(RefLib, Conversation, ParamList, 1001, BoolRes);

AstaConnection

function AstaConnSetAuthorizationOff(refNum: UInt16; hConnInstance: Pointer): Err;

Description
Turns off any Authentication information being sent to the AstaSkyWire server on connect.

Example

```
AstaConnSetAuthorizationOff(RefLib,Conversation);
```

AstaConnection

function AstaConnSetAuthorizationOn(refNum: UInt16; hConnInstance: Pointer; UserName: PChar; Password: PChar; SecurePassword: boolean): Err;

Description
Sets the Username and password to be sent to the remote ASTA SkyWire server to authenticate User Information.

Example

```
AstaConnSetAuthorizationOn(RefLib,Conversation,'ASTA','Password',False);
```

AstaConnection

function AstaConnSetCompressionOff(refNum: UInt16; hConnInstance: Pointer): Err;

Description
Sets Compression off. Requires AstaZLib.prc

Example
```
AstaConnSetCompressionOff(RefLib,Conversation);
```

AstaConnection

function AstaConnSetCompressionOn(refNum: UInt16; hConnInstance: Pointer; level: UInt32): Err;

Description
Sets Compression on where level is the compression factor and the higher the number the greater the compresssion but is also slower. Requires AstaZLib.prc

Example
```
AstaConnSetCompressionOn(RefLib,Conversation,9);
```

AstaConnection

function AstaConnSetEncryptionOff(refNum: UInt16; hConnInstance: Pointer): Err;

Description
Sets AstaAES Encryption off.

Example
AstaConnSetEncryptionOff(RefLib,Connection);

AstaConnection

function AstaConnSetEncryptionOn(refNum: UInt16; hConnInstance: Pointer; key: Pointer; keyMode: UInt32): Err;

Description
Sets EncryptionOn and passes in a Key that is used to encrypt and decrypt the data. The Server must be aware of this key. KeyMode signifies which direction and it should usually set to both (2).

Example
AstaConnSetEncryptionOn(`RefLib,Conversation,'MumsTheWord',2);`

AstaConnection

function AstaConnSetHttpOff(refnum: UInt16; hconnInstance: Pointer): Err

Will be available in Astalib 1.1 and necessary for ASTA Palm SOAP support along with the ASTA Palm

XML Parser

Description
Turns off http tunneling that was set with AstaConnSetHttpOn.

AstaConnection

function AstaConnSetHttpOn(refnum: UInt16; hConnInstance: Pointer; Address: PChar; Port: Word; Page: PChar;
Authenticate: Boolean; UserName, Password: Pchar; UseProxy: Boolean; ProxyAddress: PChar;
ProxyPort: Word; ProxyAuthenticate: Boolean; ProxyUserName: Pchar; ProxyPassWord: Pchar): Err

Will be available in Astalib 1.1 and necessary for ASTA Palm SOAP support along with the ASTA Palm XML Parser

Description
Uses http instead of tcp/ip to allow access through firewalls and through a WebServer using AstaHttp.dll.

Address:Pchar Address of the ASTA Server
Port: Word : Port of the ASTA Server
Authenticate:Boolean': Authenticate at the server
UserName:Pchar UserName
Password: Pchar; Password
UseProxy: Boolean Use a proxy Server
ProxyAddress: PChar; Address of the Proxy Server
ProxyPort: Word Port of the Proxy Server
ProxyAuthenticate: Boolean Autentication required at the Proxy
ProxyUserName: Pchar Username used by Proxy Server
ProxyPassWord: Pchar Password used by Proxy Server

Example

AstaConnection

function AstaConnSetServerVersion(refNum: UInt16; hConnInstance: Pointer; Version: UInt32): Err;

Description
Asta pda libraries can support ASTA for Windows and AstaInterOp for Linux and Windows. This swtich sets the ASTA library.

Valid values are

```
Const
Asta    = 2
AstaIO  = 3;
```

Example

```
AstaConnSetServerVersion(reflib,Connection,Asta);

AstaConnSetServerVersion(reflib,Connection,AstaIO);
```

AstaConnection

function AstaConnSetTimeout(refNum: UInt16; hConnInstance: Pointer; Timeout: UInt32): Err;

Description
Sets the value in Milliseconds that an AstaConnection will "block" in waiting for a response to the server. Since there is no threading on the Palm, sockets must connect to a server and then block or wait.

Example
AstaConnSetTimeout(reflib,Connection,5000);

AstaConnection

function AstaRequestUpdate(refNum: UInt16; hConnInstance: Pointer; Buffer: Pointer; BufLen: UInt32; **var** Result: boolean): Err;

Description

AstaConnection

function AstaSetTerminateConn(refNum: UInt16; hConnInstance: Pointer; value: boolean): Err;

Description

This will the AstaConnection to terminate the connection with the server after each connection. The default is to set this to true and is recommended over any wireless connection. With WinCE clients they may stay connected.

Example

AstaSetTerminateConn(reflib,Connection,True);

The AstaDataList is an efficient DataStructure that handles rows and columns much like a database table. To request a DataList from a remote server you must create an AstaParamList and fill it with the following values:

AstaPLSetString(AstaLibRef, params, 0,'Select * from Customer', "PalmSelect");
AstaPLSetLongInt(AstaLibRef, params, 1, 10000, "RowCount");

Param 0: the Select SQL as a String
Param 1 : the maximum rowcount

The DataList basically contains a List of ParamLists that contain data natively, stored very efficiently, wthout duplicating paramnames, like the AstaNestedParamList does. Each row can be retrieved into a

ParamList with the option of bring the Param or "FieldNames" also by calling AstaDLGetAsParamList

AstaDLCreate
AstaDLDestroy
AstaDLGetAsParamList
AstaDLGetField
AstaDLGetFieldByName
AstaDLGetFieldByNo
AstaDLGetFieldCount
AstaDLGetFieldName
AstaDLGetFieldNameC
AstaDLGetFieldNo
AstaDLGetFieldOptions
AstaDLGetFieldType
AstaDLGetIndexOfField
AstaDLGetRecordsCount
AstaDLIsEmpty

AstaDataList

function AstaDLCreate(refNum: UInt16; Buffer: Pointer; BufLen: UInt32; **var** Result: Pointer): Err;

Description
A DataList is created on a remote ASTA server in response to a select SQL . PdaMultiSQLSelect, or custom result set being streamed back to an ASTA Pda client.

AstaDLCreate Creates a DataList from a paramList AstaParamValue streamed over the wire as a Blob Param.

Example
This examples assumes the ParamList has been received using AstaConnReciveParamList. The ReturnParam:Pointer below is an AstaParamValue.

```
AstaPLGetParameter(refLib, ParamList, 0, ReturnParam);

DataPtr :=nil;
DataLen := 0;
//with Dataptr as NIL the size of the blob is returned in DataLen
AstaPLParamGetAsBlob(RefLib, ReturnParam, DataPtr, DataLen);

DataPtr := MemPtrNew(DataLen+1);//allocate some memory now

AstaPLParamGetAsBlob(Reflib, ReturnParam, DataPtr, DataLen);

DataList:=nil;

AstaDLCreate(Reflib, DataPtr, DataLen, DataList);

MemPtrFree(DataPtr);
```

AstaDataList

function AstaDLDestroy(refNum: UInt16; hDLInstance: Pointer): Err;

Description
Destroys an AstaDataList and frees up it's resources;

Example
AstaDLDestroy(refLib,DataList);

AstaDataList

function AstaDLGetAsParamList(refNum: UInt16; hDLInstance: Pointer; recordNumber: UInt32; putNames: boolean; **var** Result: Pointer): Err;

Description
Retrieves an AstaParamList at the row designated as RecordNumber into the Var Result:Pointer. The number of rows or records can be obtained by calling AstaDLGetRecordsCount.

Example
AstaDLGetAsParamList(refLib,DataList,1,True,TempParamList);

AstaDataList

function AstaDLGetField(refNum: UInt16; hDLInstance: Pointer; **Index**: UInt32; **var** Result: Pointer): Err;

Description
Returns a handle to a Field by a given Index.

AstaDataList

function AstaDLGetFieldByName(refNum: UInt16; hDLInstance: Pointer; **Name**: PChar; **var** Result: Pointer): Err;

Description
Returns the "handle" to the "current" field by given name

AstaDataList

function AstaDLGetFieldByNo(refNum: UInt16; hDLInstance: Pointer; **Index**: UInt32; **var** Result: Pointer): Err;

Description

Returns the "handle" to the field by given index

AstaDataList

function AstaDLGetFieldCount(refNum: UInt16; hDLInstance: Pointer; **var** Result: UInt32): Err;

Description
Returns the number of fields in a DataLis in the Var Result: UInt32.

Example
AstaDLGetFieldCount(refLib,DataList,Fieldcount);

AstaDataList

function AstaDLGetFieldName(refNum: UInt16; hDLInstance: Pointer; hFieldInstance: Pointer; NameBuf: PChar; **var** BufLen: UInt32): Err;

Description
Returns the fieldname of a given field when a Field Handle is passed in as well as the length of the field in BufLen

AstaDataList

function AstaDLGetFieldNameC(refNum: UInt16; hDLInstance: Pointer; hFieldInstance: Pointer; NameBuf: PChar): Err;

Description
Returns the name of the field

AstaDataList

function AstaDLGetFieldNo(refNum: UInt16; hDLInstance: Pointer; hFieldInstance: Pointer; **var** Result: UInt32): Err;

Description
Returns the Nnumber of the field.

AstaDataList

function AstaDLGetFieldOptions(refNum: UInt16; hDLInstance: Pointer; hFieldInstance: Pointer; **var** Result: UInt32): Err;

```
#define pfaHiddenCol    1
#define pfaReadonly     2
#define pfaRequired     4
```

```
#define pfaLink          8
#define pfaUnNamed       16
#define pfaFixed                 3
```

Description
Returns the options of the field.

AstaDataList

function AstaDLGetFieldType(refNum: UInt16; hDLInstance: Pointer; hFieldInstance: Pointer; **var** Result: AstaFieldType): Err;

Description
Returns the type of the field for a given Field Handle

AstaDataList

function AstaDLGetIndexOfField(refNum: UInt16; hDLInstance: Pointer; hFieldInstance: Pointer; **var** Result: UInt32): Err;

Description
Returns the "index" of the field in Result of a given Field Handle.

AstaDataList

function AstaDLGetRecordsCount(refNum: UInt16; hDLInstance: Pointer; **var** Result: UInt32): Err;

Description
Returns the number of records contained in an AstaDataList in Result. This count usually corresponds to the number of rows requested to be returned from a remote server.

Example

```
AstaDLGetRecordsCount(reflib,DataList,RecordCount);
```

AstaDataList

function AstaDLIsEmpty(refNum: UInt16; hDLInstance: Pointer; **var** Result: boolean): Err;

Description

If Result then the AstaDataList is Empty.

The AstaNestedParamList provides a datatructure that works like a DataSet with a row/column abstraction in order to handle SQL result requests from ASTA SkyWire servers with all the data stored as Strings. This makes it easy to use with User Interface Controls. To handle data natively, and a bit more efficiently use DataLists. NestedParamLists are actually an AstaParamList of AstaParamLists with the first ParamList a list of Fields and the rest of the paramlists containing the actual SQL Result sets.

Nested Param List Example

AstaNPLCreate
ASTANPLDestroy
AstaNPLGetField
AstaNPLGetFieldByName
AstaNPLGetFieldCount
AstaNPLGetRecordsCount
AstaNPLIsBOF
AstaNPLisEmpty
AstaNPLIsEOF
AstaNPLNext
AstaNPLPrevious

AstaNestedParamList

function AstaNPLCreate(refNum: UInt16; hPLInstance: Pointer; **var** Result: Pointer): Err;

Description
Creates an AstaNestedParamList in Result from an incoming ParamList from the server in response to PdaToken PdaNestedSQLSelect (1010)

Example

```
AstaNPLCreate(RefLib, ParamList, NPL);
```

AstaNestedParamList

function AstaNPLDestroy(refNum: UInt16; hPLInstance: Pointer): Err;

Description
Disposes of an AstaNestedfParamList previously created with AstaNPLCreate and releases it's resources.

Example

```
AstaNPLDestroy(RefLib, NPL);
```

AstaNestedParamList

function AstaNPLGetField(refNum: UInt16; hPLInstance: Pointer; **Index**: UInt32; **var** Result: Pointer): Err;

Description
Retrieves a Field as an AstaParamValue from a NestedParamList.

Example

```
for i := 0 to ParamCount - 1 do
    begin
  Value := nil;
  ValueLen := 0;
  AstaNPLGetField(RefLib, NPL, i, Fld);
  if (fld <> nil) then
  begin
    AstaPLParamGetAsString(RefLib, fld, Value, ValueLen);
    Inc(ValueLen);
    Value := MemPtrNew(ValueLen);
    AstaPLParamGetAsString(RefLib, fld, Value, ValueLen);
    Params[i] := Value;
  end;
 end;
```

AstaNestedParamList

function AstaNPLGetFieldByName(refNum: UInt16; hPLInstance: Pointer; **Name**: PChar; **var** Result: Pointer): Err;

Description

Retrieves an AstaNestedParamList Field byName into Result.

Example

AstaNestedParamList

function AstaNPLGetFieldCount(refNum: UInt16; hPLInstance: Pointer; **var** Result: UInt32): Err;

Description
Returns the number of Fields in a AstaNestedParamList

Example
AstaNPLGetFieldCount(RefLib, NPL, ParamCount);

AstaNestedParamList

function AstaNPLGetRecordsCount(refNum: UInt16; hPLInstance: Pointer; **var** Result: **Integer**): Err;

Description

Returns the number of rows in an AstaNestedParamList where each row is an AstaParamList

Example
AstaNPLGetRecordsCount(RefLib, NPL, Count);

AstaNestedParamList

function AstaNPLIsBOF(refNum: UInt16; hPLInstance: Pointer; **var** Result: boolean): Err;

Description

Returns whether the postion of the AstaNestedParamList is Beginning of File similar to the TDataSet.BOF Call

AstaNestedParamList

function AstaNPLIsEmpty(refNum: UInt16; hPLInstance: Pointer; **var** Result: boolean): Err;

Description
Checks to see if an AstaNestedParamList is Empty and returns a boolean in Var Result if it is.

Example

AstaNPLIsEmpty(RefLib, NPL, BoolRes);

AstaNestedParamList

function AstaNPLIsEOF(refNum: UInt16; hPLInstance: Pointer; **var** Result: boolean): Err;

Description

Returns whether the postion of the AstaNestedParamList is End of File similar to the TDataSet.EOF Call

Example

AstaNestedParamList

function AstaNPLNext(refNum: UInt16; hPLInstance: Pointer; **var** Result: boolean): Err;

Description

Moves to the next embedded AstaParamList from an AstaNestedParamlist

AstaNestedParamList

function AstaNPLPrevious(refNum: UInt16; hPLInstance: Pointer; **var** Result: boolean): Err;

Description

Moves to the previous embedded AstaParamList from an AstaNestedParamlist

The ASTA Unified Messaging Initiative allows for cross platform messaging by using AstaParamLists which are easy to use and fully streamable Data containers that can be transported between ASTA clients and servers and between ASTA clients regardless of hardware or operating system.

AstaPLDestroy
AstaPLClear
AstaPLCopyTo
AstaPLCreate
AstaPLCreateTransportList
AstaPLGetCount
AstaPLGetParameter
AstaPLGetParameterByName
AstaPLGetParameterIndex
AstaPLSetBlob
AstaPLSetBoolean
AstaPLSetByte
AstaPLSetDate
AstaPLSetDateTime
AstaPLSetDouble

AstaPLSetFromTransportList
AstaPLSetLongInt
AstaPLSetNull
AstaPLSetSingle
AstaPLSetSmallInt
AstaPLSetString
AstaPLSetTime

AstaParamList

function AstaPLClear(refNum: UInt16; hPLInstance: Pointer): Err;

Description

Clears the ParamList of all items.

Example

AstaPLClear(reflib,AstaParamList);

AstaParamList

function AstaPLCopyTo(refNum: UInt16; hPLInstance: Pointer; hDestPLInstance: Pointer): Err;

Description

Copes one AstaParamList to another.

AstaParamList

function AstaPLCreate(refNum: UInt16; InitialSize: Int32; **var** Result: Pointer): Err;

Description

Creates an AstaParamList

Example
This creates an AstaParamlist with 5 initial ParamItems.

```
var
 ParamList:Pointer;

AstaPLCreate(reflib,5,ParamList);
```

AstaParamList

function AstaPLCreateTransportList(refNum: UInt16; hPLInstance: Pointer; **var** Buffer: Pointer; **var** BufLen: **integer**): Err;

Description
Used internally by AstaLIb.prc to stream an AstaParamList.

AstaParamList

function AstaPLDestroy(refNum: UInt16; hPLInstance: Pointer): Err;

Description

Destroys an AstaParamList and releases resources.

Example

AstaParamList

function AstaPLGetCount(refNum: UInt16; hPLInstance: Pointer; **var** Result: UInt32): Err;

Description

Returns the number of items in the AstaParamList

Example
AstaPLGetCount(reflib,ParamList,Count);

AstaParamList

function AstaPLGetParameter(refNum: UInt16; hPLInstance: Pointer; **Index**: UInt32; **var** Result: Pointer): Err;

Description

Retrieves an AstaParamValue at Index from an AstaParamList

Example
var
ParamList : pointer;
ParamItem:Pointer;

AstaPlGetParameter(RefLib, ParamList, 0, ParamItem);

AstaParamList

function AstaPLSetBlob(refNum: UInt16; hPLInstance: Pointer; **Index**: UInt32; value: Pointer; ValueLen: UInt32; **Name**:

PChar): Err;

Description

Sets an AstaParamValue with a buffer of Value of Length ValueLen.

Example

AstaPlSetBlob(RefLib,Params,0,TheBlob,BlobSize,'Blob');

AstaParamList

function AstaPLSetBoolean(refNum: UInt16; hPLInstance: Pointer; **Index**: UInt32; value: boolean; **Name**: PChar): Err;

Description

Sets an AstaParamValue at Index to to a Boolean value with ParamName Name.

AstaParamList

function AstaPLSetByte(refNum: UInt16; hPLInstance: Pointer; **Index**: UInt32; Value: char; **Name**: PChar): Err;

Description

Sets an AstaParamValue at Index to a Byte value with ParamName Name.

Example

AstaParamList

function AstaPLSetDate(refNum: UInt16; hPLInstance: Pointer; **Index**: UInt32; Value: DateTimePtr; **Name**: PChar): Err;

Description

Sets an AstaParamValue at Index to a DateType Value with ParamName Name.

Example

AstaParamList

function AstaPLSetDateTime(refNum: UInt16; hPLInstance: Pointer; **Index**: UInt32; Value: DateTimePtr; **Name**: PChar): Err;

Description

Sets an AstaParamValue at Index to a DateTime value with ParamName Name.

AstaParamList

function AstaPLSetDouble(refNum: UInt16; hPLInstance: Pointer; **Index**: UInt32; value: double; **Name**: PChar): Err;

Description

Sets a Param Item at Index to to a double value with ParamName Name.

Example
AstaPlSetDouble(RefLib,Params,1,5.44,'Double');

AstaParamList

function AstaPLSetFromTransportList(refNum: UInt16; hPLInstance: Pointer; Buf: Pointer; BufLen: UInt32): Err;

Description
Used internally to receive a raw buffer and convert to an AstaParamList.

AstaParamList

function AstaPLSetLongInt(refNum: UInt16; hPLInstance: Pointer; **Index**: UInt32; value: Int32; **Name**: PChar): Err;

Description
Sets an AstaParamValue at Index to LongInt with a ParamName of Name.

Example

AstaPlSetLongInt(RefLib,Params,1,45,'Age');

AstaParamList

function AstaPLSetNull(refNum: UInt16; hPLInstance: Pointer; **Index**: UInt32; fType: AstaFieldType; **Name**: PChar): Err;

Description

Sets an AstaParamValue to Null.

Example
AstaPLSetNull(RefLib,Params,1,pftString,'Department');

AstaParamList

function AstaPLSetSingle(refNum: UInt16; hPLInstance: Pointer; **Index**: UInt32; value: single; **Name**: PChar): Err;

Description
Sets a Param Item at Index to to a Single value with ParamName Name.

Example

AstaParamList

function AstaPLSetSmallInt(refNum: UInt16; hPLInstance: Pointer; **Index**: UInt32; value: Int32; **Name**: PChar): Err;

Description

Sets an AstaParamValue at Index to a SmallInt value with ParamName Name.

Example

AstaParamList

function AstaPLSetString(refNum: UInt16; hPLInstance: Pointer; **Index**: UInt32; Value: PChar; **Name**: PChar): Err;

Description

Sets a String AstaParamValue at index with ParamName of Name

Example

AstaPlSetString(RefLib, ParamList, 0, 'Select * from Customers', 'PalmSelect');

AstaParamList

function AstaPLSetTime(refNum: UInt16; hPLInstance: Pointer; **Index**: UInt32; Value: DateTimePtr; **Name**: PChar): Err;

Description

Sets a Time AstaParamValue at index with ParamName of Name

Example

AstaParamValues are the items contained in the expandable AstaParamLists explained more fully in the ASTA Unified Messaging Initiative

Parameter name - the name of the parameter
Parameter type - the mode of the parameter, in, out, inout, result.
Data type - AstaFieldType which follows the database types (char, bit, integer, etc.)
IsNull flag - a flag indicating if the data is null (a bit representing true or false)
Bytes of data - the actual data as a list of bytes

AstaPLparamGetAsBlob
AstaPLParamGetAsBoolean
AstaPLParamGetAsDate
AstaPLParamGetAsDateTime
AstaPLParamGetAsDouble
AstaPLParam.GetAsLongInt
ASTAPLParamGetAsString
AstaPLParamGetAsStringC
AstaPLParamGetAsTime
AstaPLParamGetName
AstaPLParamGetType
AstaPLParamGetIsNull
AstaPLParamSetName

AstaFieldType=(pftUnknown, pftByte, pftSmallint, pftLongint, pftBoolean, pftSingle, pftDouble, pftDate, pftTime, pftDateTime, pftString, pftBlob);
Description

Value	**Meaning**
pftUnknown	
pftByte	
pftSmallint	
pftLongint	
pftBoolean	
pftSingle	
pftDouble	
pftDate	
pftTime	
pftDateTime	
pftString	
pftBlob	

AstaParamValue

function AstaPLGetParameterByName(refNum: UInt16; hPLInstance: Pointer; **Name**: PChar; **var** Result: Pointer): Err;

Description

Retrieves an AstaParamValue by ParamName from an AstaParamList

Example

AstaParamValue

function AstaPLGetParameterIndex(refNum: UInt16; hPLInstance: Pointer; hParamInstance: Pointer; **var** Result: UInt32): Err;

Description

Returns the Index of an AstaParamValue from an AstaParamList

Example

AstaParamValue

function AstaPLParamGetAsBlob(refNum: UInt16; hInstance: Pointer; Buffer: Pointer; **var** BufLen: UInt32): Err;

Description

Copies an AstaParamValue Data into Buffer and returns the size of the Blob in BufLen.

AstaParamValue

function AstaPLParamGetAsBoolean(refNum: UInt16; hInstance: Pointer; **var** Result: boolean): Err;

Description

Retrieves the Boolean value of an AstaParamValue in Result:Boolean.

AstaParamValue

function AstaPLParamGetAsDate(refNum: UInt16; hInstance: Pointer; Date: DateTimePtr; **var** Result: boolean): Err;

Description

Retrieves the DateTimePtr value of an AstaParamValue in Date and if successful returns True in Result.

AstaParamValue

function AstaPLParamGetAsDateTime(refNum: UInt16; hInstance: Pointer; Time: DateTimePtr; **var**

Result: boolean): Err;

Description

Retrieves the DateTimePtr value of an AstaParamValue in Time and if successful returns True in Result.

AstaParamValue

function AstaPLParamGetAsDouble(refNum: UInt16; hInstance: Pointer; **var** Result: double): Err;

Description

Retrieves a double value of an AstaParamValue in Result.

AstaParamValue

function AstaPLParamGetAsLongInt(refNum: UInt16; hInstance: Pointer; **var** Result: Int32): Err;

Description

Retrieves a LongInt value of an AstaParamValue in Result.

AstaParamValue

function AstaPLParamGetAsString(refNum: UInt16; hInstance: Pointer; NameBuf: PChar; **var** BufLen: UInt32): Err;

Description
Retrieves the AstaParamValue into NameBuf and the length of the Pchar into BufLen.

AstaParamValue

function AstaPLParamGetAsStringC(refNum: UInt16; hInstance: Pointer; NameBuf: PChar): Err;

Description
Retrieves the AstaParamValue into NameBuf

AstaParamValue

function AstaPLParamGetAsTime(refNum: UInt16; hInstance: Pointer; Time: DateTimePtr; **var** Result: boolean): Err;

Description

AstaParamValue

function AstaPLParamGetIsNull(refNum: UInt16; hInstance: Pointer; **var** Result: boolean): Err;

Description
Returns True to Result of the if the ParamValue is null.

AstaParamValue

function AstaPLParamGetName(refNum: UInt16; hInstance: Pointer; NameBuf: PChar; **var** BufLen: UInt32): Err;

Description
Retrieves the Name of the ParamValue along with the buffer length.

AstaParamValue

function AstaPLParamGetNameC(refNum: UInt16; hInstance: Pointer; NameBuf: PChar): Err;

Description
Retrieves the Name of the ParamValue.

Example

AstaParamValue

function AstaPLParamGetType(refNum: UInt16; hInstance: Pointer; **var** Result: AstaFieldType): Err;

Description

Retrieves a AstaFieldtype value of an AstaParamValue in Result.

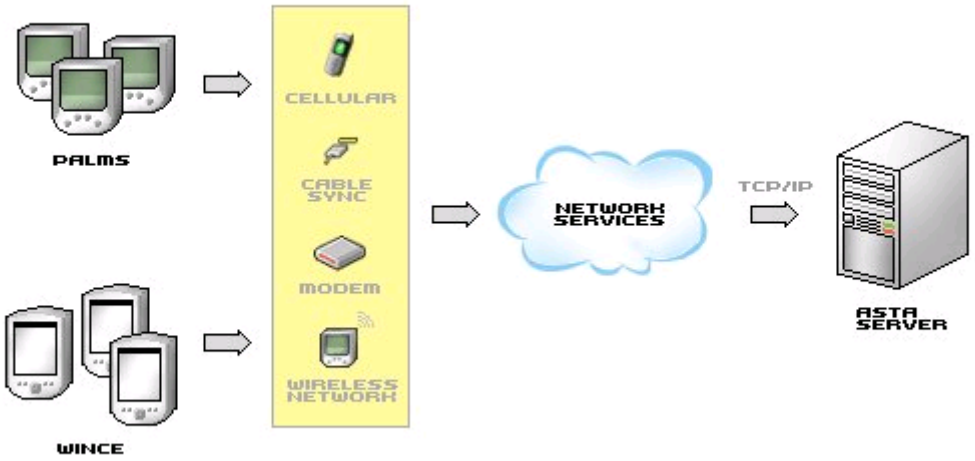
AstaParamValue

function AstaPLParamSetName(refNum: UInt16; hInstance: Pointer; Value: PChar): Err;

Description

Sets an AstaParamValue Name to Value

1.10.2.4.3.2 CodeWarrior



The ASTA SkyWire architecture consists of remote thin clients, connecting to a middle tier Application Server via TCP/IP. SkyWire Servers are available using a number of languages include Visual Basic, Delphi and Java. This document describes how to use CodeWarrior Clients to access remote SkyWire Servers servers and perform standard tasks using the SkyWire component.

ASTA Skywire consists of a middleware Server and client side components to access that server. The following components are used by SkyWire clients to perform standard tasks.

AstaConnection
AstaParamList
AstaParamValue
AstaDataList
AstaNestedParamList

The AstaClientConnection is the client end of the TCP/IP connection. It connects the ASTA client to the ASTA SkyWire Server. The Address and Port properties on the ASTA client must match the Address and Port properties on the ASTA SkyWire Server. Typically the handheld device would have a form where the IPAddress and Port can be input by the user so that the AstaClientConnection can connect to a SkyWire server and authenticate using a Username and Password.

AstaConnClose
AstaConnCreate
AstaConnDestroy
AstaConnGetFile
AstaConnGetLastError
AstaConnGetUserError
AstaConnOpen
AstaConnReceiveParamList
AstaConnSendParamList
AstaConnSetAuthenticationOff
AstaConnSetAuthenticationOn
AstaConnSetCompressionOff
AstaConnSetCompressionOn
AstaConnSetEncryptionOff
AstaConnSetEncryptionOn
AstaConnSetHttpOn
AstaConnSetHttpOff
AstaConnSetTimeout
AstaRequestUpdate

AstaConnection

```
Err AstaConnClose(UInt refNum, VoidPtr hConnInstance, bool *Result)
```

Description

Closes a connection opened after calling AstaConnOpen.

Example

```
AstaConnClose(RefLib, Conversation, BoolRes);
```

AstaConnection

```
Err AstaConnCreate(UInt refNum, VoidPtr *hConnInstance)
```

Description
Creates an AstaConnection as the Var Result:Pointer where refNum represents the handle to the Asta Shared Library.

Example
```
AstaConnCreate(RefLib, Conversation);
```

AstaConnection

```
Err AstaConnDestroy(UInt refNum, VoidPtr hConnInstance)
```

Description
Destroys and AstaConnection and releases it's resources that has been created with AstaConnOpen.

Example
```
AstaConnDestroy(RefLib, Conversation);
```

AstaConnection

```
Err AstaConnGetFile(UInt refNum, VoidPtr hConnInstance, ULONG BlockSize, char* Request, char*
LocalFileName, bool *Result)
```

Description
Retrieves a remote file from an ASTA Server in BlockSize "chunks". This allows, for example, for large images to be streamed down efficiently.

AstaConnection

```
Err AstaConnGetLastError(UInt refNum, VoidPtr hConnInstance, Int32 *Result)
```

Description
Returns the last error generated by the AstaClientConnection. AstaSky Wire servers are codedso as to return an Integer value on each client request where a non-zero value is considered to be an error.

AstaConnection

```
Err AstaConnGetUserError(UInt refNum, VoidPtr hConnInstance, Int32 *Result)
```

escription
Returns the last error generated by the AstaClientConnection. AstaSky Wire servers are coded so as to return an Integer value on each client request where a non-zero value is considered to be an error. Typically errors are also returned with the server side error message or exception in an AstaParamValue of Name "Error"

AstaConnection

```
Err AstaConnOpen(UInt refNum, VoidPtr hConnInstance,
                          CharPtr Address, Word Port, bool *Result)
```

Description
Attempts to connect to a remote AstaSkyWire server using the Address and Port and will return a

False a connection to the server cannot be achieved. No data is transferred to the server. The Asta SkyWire shared library must already be loaded.

Example

```
AstaConnOpen(RefLib, Conversation, '127.0.01',9050, BoolRes);
```

AstaConnection

```
Err AstaConnReceiveStream(UInt refNum, VoidPtr hConnInstance,
                          CharPtr* Buffer, ULONG *BufSize, bool *Result)
```

Description

Receives an AstaParamList from a remote ASTA Server where the hPLIntance is an AstaParamList that has been created with a call to AstaPLCreate. If the ParamList was sucessfully retrieved from the server the Var Result:Boolean would be set to True.

Example

Typically, if you had an existing Paramlist that you had used in sending params to a remote server you would want to clear it first.

```
AstaPlClear(RefLib, ParamList);
AstaConnReceiveParamList(RefLib, Conversation, ParamList, BoolRes);
```

AstaConnection

```
Err AstaConnSendParamList(UInt refNum, VoidPtr hConnInstance,
                          VoidPtr hPLInstance, ULONG MsgToken, bool *Result)
```

Description

Sends an AstaParam to a remote Server. The AstaParamList must have been created and typically is filled with values before being sent to the remote server. The MsgToken:UInt32 will determine how the server processes the request. If the ParamList was successfully sent to the server, teh Var Result:Boolean will be set to true.

Example

AstaConnSendParamList(RefLib, Conversation, ParamList, 1001, BoolRes);

AstaConnection

```
Err AstaConnSetAuthorizationOff(UInt refNum, VoidPtr hConnInstance)
```

Description

Turns off any Authentication information being sent to the AstaSkyWire server on connect.

Example

```
AstaConnSetAuthorizationOff(RefLib,Conversation);
```

AstaConnection

```
Err AstaConnSetAuthorizationOn(UInt refNum, VoidPtr hConnInstance, CharPtr UserName, CharPtr
Password, bool SecurePassword)
```

Description
Sets the Username and password to be sent to the remote ASTA SkyWire server to authenticate User Information.

Example
```
AstaConnSetAuthorizationOn(RefLib,Conversation,'ASTA','Password',False);
```

AstaConnection

```
Err AstaConnSetCompressionOff(UInt refNum, VoidPtr hConnInstance)
```

Description
Sets Compression off. Requires AstaZLib.prc

Example
```
AstaConnSetCompressionOff(RefLib,Conversation);
```

AstaConnection

```
Err AstaConnSetCompressionOn(UInt refNum, VoidPtr hConnInstance, Int32 level)
```

Description
Sets Compression on where level is the compression factor and the higher the number the greater the compresssion but is also slower. Requires AstaZLib.prc

Example
```
AstaConnSetCompressionOn(RefLib,Conversation,9);
```

AstaConnection

```
Err AstaConnSetEncryptionOff(UInt refNum, VoidPtr hConnInstance)
```

Description
Sets AstaAES Encryption off.

Example
AstaConnSetEncryptionOff(RefLib,Connection);

AstaConnection

```
Err AstaConnSetEncryptionOn(UInt refNum, VoidPtr hConnInstance, VoidPtr key, Int32 keyMode)
```

Description
Sets EncryptionOn and passes in a Key that is used to encrypt and decrypt the data. The Server must be aware of this key. KeyMode signifies which direction and it should usually set to both (2).

Example
AstaConnSetEncryptionOn(`RefLib,Conversation,'MumsTheWord',2);`

AstaConnection

function AstaConnSetHttpOff(refnum: UInt16; hconnInstance: Pointer): Err

Will be available in Astalib 1.1 and necessary for ASTA Palm SOAP support along with the ASTA Palm XML Parser

Description
Turns off http tunneling that was set with AstaConnSetHttpOn.

AstaConnection

function AstaConnSetHttpOn(refnum: UInt16; hConnInstance: Pointer; Address: PChar; Port: Word; Page: PChar;
Authenticate: Boolean; UserName, Password: Pchar; UseProxy: Boolean; ProxyAddress: PChar;
ProxyPort: Word; ProxyAuthenticate: Boolean; ProxyUserName: Pchar; ProxyPassWord: Pchar): Err

Will be available in Astalib 1.1 and necessary for ASTA Palm SOAP support along with the ASTA Palm XML Parser

Description
Uses http instead of tcp/ip to allow access through firewalls and through a WebServer using AstaHttp.dll.

Address:Pchar Address of the ASTA Server
Port: Word : Port of the ASTA Server
Authenticate:Boolean': Authenticate at the server
UserName:Pchar UserName
Password: Pchar; Password
UseProxy: Boolean Use a proxy Server
ProxyAddress: PChar; Address of the Proxy Server
ProxyPort: Word Port of the Proxy Server
ProxyAuthenticate: Boolean Autentication required at the Proxy
ProxyUserName: Pchar Username used by Proxy Server
ProxyPassWord: Pchar Password used by Proxy Server

Example

AstaConnection

```
Err AstaConnSetServerVersion(UInt refNum, VoidPtr hConnInstance, ULONG version)
```

Description
Asta pda libraries can support ASTA for Windows and AstaInterOp for Linux and Windows. This swtich sets the ASTA library.

Valid values are

Const
Asta = 2
AstaIO = 3;

Example
AstaConnSetServerVersion(reflib,Connection,Asta);

AstaConnSetServerVersion(reflib,Connection,AstaIO);

AstaConnection

```
Err AstaConnSetTimeout(UInt refNum, VoidPtr hConnInstance, Int32 Timeout)
```

Description
Sets the value in Milliseconds that an AstaConnection will "block" in waiting for a response to the server. Since there is no threading on the Palm, sockets must connect to a server and then block or wait.

Example
AstaConnSetTimeout(reflib,Connection,5000);

AstaConnection

```
Err AstaRequestUpdate(UInt refNum, VoidPtr hConnInstance, VoidPtr Buffer, Int32 BufLen, Int32
*Result)
```

Description
As part of the ASTA Hitchhikers Guide to the Wireless Universe, SkyWire clients have the ability to update themselves from remote servers. This call requests an update from a remote server.

AstaConnection

```
Err AstaSetTerminateConn(UInt refnum, VoidPtr hInstance, bool value)
```

Description

This will the AstaConnection to terminate the connection with the server after each connection. The default is to set this to true and is recommended over any wireless connection.

The AstaDataList is an efficient DataStructure that handles rows and columns much like a database table. To request a DataList from a remote server you must create an AstaParamList and fill it with the following values:

AstaPLSetString(AstaLibRef, params, 0,'Select * from Customer', "PalmSelect");
AstaPLSetLongInt(AstaLibRef, params, 1, 10000, "RowCount");

Param 0: the Select SQL as a String
Param 1 : the maximum rowcount

The DataList basically contains a List of ParamLists that contain data natively, stored very efficiently, wthout duplicating paramnames, like the AstaNestedParamList does. Each row can be retrieved into a ParamList with the option of bring the Param or "FieldNames" also by calling AstaDLGetAsParamList

AstaDLCreate
AstaDLDestroy
AstaDLGetAsParamList
AstaDLGetField
AstaDLGetFieldByName
AstaDLGetFieldByNo
AstaDLGetFieldCount
AstaDLGetFieldName
AstaDLGetFieldNameC
AstaDLGetFieldNo
AstaDLGetFieldOptions
AstaDLGetFieldType
AstaDLGetIndexOfField
AstaDLGetRecordsCount
AstaDLIsEmpty

AstaDataList

```
Err AstaDLCreate(UInt refNum, VoidPtr Buffer, Int32 BufLen, VoidPtr *hDLInstance)
```

Description
A DataList is created on a remote ASTA server in response to a select SQL . PdaMultiSQLSelect, or custom result set being streamed back to an ASTA Pda client.

AstaDLCreate Creates a DataList from a paramList AstaParamValue streamed over the wire as a Blob Param.

AstaDataList

```
Err AstaDLDestroy(UInt refNum, VoidPtr hDLInstance)
```

Description
Destroys an AstaDataList and frees up it's resources;

Example
AstaDLDestroy(refLib,DataList);

AstaDataList

```
Err AstaDLGetAsParamList(UInt refNum, VoidPtr hDLInstance, unsigned long recordNumber, bool
putNames, VoidPtr *hPLInstance)
```

Description
Retrieves an AstaParamList at the row designated as RecordNumber into the Var Result:Pointer. The number of rows or records can be obtained by calling AstaDLGetRecordsCount.

Example
AstaDLGetAsParamList(refLib,DataList,1,True,TempParamList);

AstaDataList

```
Err AstaDLGetField(UInt refNum, VoidPtr hDLInstance, UInt32 Index, VoidPtr *hInstance)
```

Description
Returns a handle to a Field by a given Index.

AstaDataList

```
Err AstaDLGetFieldByName(UInt refNum, VoidPtr hDLInstance, CharPtr Name, VoidPtr *hInstance)
```

Description
Returns the "handle" to the "current" field by given name

AstaDataList

```
Err AstaDLGetFieldByNo(UInt refNum, VoidPtr hDLInstance, UInt32 Index, VoidPtr *hInstance)
```

Description
Returns the "handle" to the field by given index

AstaDataList

```
Err AstaDLGetFieldCount(UInt refNum, VoidPtr hDLInstance, long *Result)
```

Description
Returns the number of fields in a DataLis in the Var Result: UInt32.

Example
AstaDLGetFieldCount(refLib,DataList,Fieldcount);

AstaDataList

```
Err AstaDLGetFieldName(UInt refNum, VoidPtr hDLInstance, VoidPtr  hFieldInstance, CharPtr
NameBuf, Int32 *BufLen)
```

Description
Returns the fieldname of a given field when a Field Handle is passed in as well as the length of the field in BufLen

AstaDataList

```
Err AstaDLGetFieldNameC(UInt refNum, VoidPtr hDLInstance, VoidPtr hFieldInstance, CharPtr
NameBuf)
```

Description
Returns the name of the field

AstaDataList

```
Err AstaDLGetFieldNo(UInt refNum, VoidPtr hDLInstance, VoidPtr hFieldInstance, UInt32 *Result)
```

Description
Returns the Nnumber of the field.

AstaDataList

```
Err AstaDLGetFieldOptions(UInt refNum, VoidPtr hDLInstance, VoidPtr hFieldInstance, UInt32
*Result)
```

```
#define pfaHiddenCol        1
#define pfaReadonly         2
#define pfaRequired         4
#define pfaLink             8
#define pfaUnNamed          16
#define pfaFixed                3
```

Description
Returns the options of the field.

AstaDataList

```
Err AstaDLGetFieldType(UInt refNum, VoidPtr hDLInstance, VoidPtr hFieldInstance, AstaFieldType
*Result)
```

Description
Returns the type of the field for a given Field Handle

AstaDataList

```
Err AstaDLGetIndexOfField(UInt refNum, VoidPtr hDLInstance, VoidPtr hFieldInstance, UInt32
*Result)
```

Description
Returns the "index" of the field in Result of a given Field Handle.

AstaDataList

```
Err AstaDLGetRecordsCount(UInt refNum, VoidPtr hDLInstance, UInt32 *Result)
```

Description
Returns the number of records contained in an AstaDataList in Result. This count usually corresponds to the number of rows requested to be returned from a remote server.

AstaDataList

```
Err AstaDLIsEmpty(UInt refNum, VoidPtr hDLInstance, bool *Result)
```

Description

If Result then the AstaDataList is Empty.

The AstaNestedParamList provides a datatructure that works like a DataSet with a row/column abstraction in order to handle SQL result requests from ASTA SkyWire servers with all the data stored as Strings. This makes it easy to use with User Interface Controls. To handle data natively, and a bit more efficiently use DataLists. NestedParamLists are actually an AstaParamList of AstaParamLists with the first ParamList a list of Fields and the rest of the paramlists containing the actual SQL Result sets.

Nested Param List Example

AstaNPLCreate
ASTANPLDestroy
AstaNPLGetField
AstaNPLGetFieldByName
AstaNPLGetFieldCount
AstaNPLGetRecordsCount
AstaNPLIsBOF
AstaNPLisEmpty
AstaNPLIsEOF
AstaNPLNext
AstaNPLPrevious

AstaNestedParamList

```
Err AstaNPLCreate(UInt refNum, VoidPtr hPLInstance, VoidPtr *hNPLInstance)
```

Description
Creates an AstaNestedParamList in Result from an incoming ParamList from the server in response to PdaToken PdaNestedSQLSelect (1010)

Example

```
AstaNPLCreate(RefLib, ParamList, NPL);
```

AstaNestedParamList

```
Err AstaNPLDestroy(UInt refNum, VoidPtr hPLInstance)
```

Description
Disposes of an AstaNestedfParamList previously created with AstaNPLCreate and releases it's resources.

Example

```
AstaNPLDestroy(RefLib, NPL);
```

AstaNestedParamList

```
Err AstaNPLGetField(UInt refNum, VoidPtr hPLInstance, UInt32 Index, VoidPtr *hInstance)
```

Description
Retrieves a Field as an AstaParamValue from a NestedParamList.

AstaNestedParamList

```
Err AstaNPLGetFieldByName(UInt refNum, VoidPtr hPLInstance, CharPtr Name, VoidPtr *hInstance)
```

Description

Retrieves an AstaNestedParamList Field byName into Result.

AstaNestedParamList

```
Err AstaNPLGetFieldCount(UInt refNum, VoidPtr hPLInstance, long *Result)
```

Description
Returns the number of Fields in a AstaNestedParamList

AstaNestedParamList

```
Err AstaNPLGetRecordsCount(UInt refNum, VoidPtr hPLInstance, long *Result)
```

Description

Returns the number of rows in an AstaNestedParamList where each row is an AstaParamList

Example
AstaNPLGetRecordsCount(RefLib, NPL, Count);

AstaNestedParamList

```
Err AstaNPLIsBOF(UInt refNum, VoidPtr hPLInstance, bool *Result)
```

Description

Returns whether the postion of the AstaNestedParamList is Beginning of File similar to the TDataSet.BOF Call

AstaNestedParamList

```
Err AstaNPLIsEmpty(UInt refNum, VoidPtr hPLInstance, bool *Result)
```

Description
Checks to see if an AstaNestedParamList is Empty and returns a boolean in Var Result if it is.

Example

AstaNPLIsEmpty(RefLib, NPL, BoolRes);

AstaNestedParamList

```
Err AstaNPLIsEOF(UInt refNum, VoidPtr hPLInstance, bool *Result)
```

Description

Returns whether the postion of the AstaNestedParamList is End of File similar to the TDataSet.EOF Call

Example

AstaNestedParamList

```
Err AstaNPLNext(UInt refNum, VoidPtr hPLInstance, bool *Result)
```

Description

Moves to the next embedded AstaParamList from an AstaNestedParamlist

AstaNestedParamList

```
Err AstaNPLPrevious(UInt refNum, VoidPtr hPLInstance, bool *Result)
```

Description

Moves to the previous embedded AstaParamList from an AstaNestedParamlist

The ASTA Unified Messaging Initiative allows for cross platform messaging by using AstaParamLists which are easy to use and fully streamable Data containers that can be transported between ASTA clients and servers and between ASTA clients regardless of hardware or operating system.

AstaPLDestroy
AstaPLClear
AstaPLCopyTo
AstaPLCreate
AstaPLCreateTransportList
AstaPLGetCount
AstaPLGetParameter
AstaPLGetParameterByName
AstaPLGetParameterIndex
AstaPLSetBlob
AstaPLSetBoolean
AstaPLSetByte
AstaPLSetDate
AstaPLSetDateTime
AstaPLSetDouble
AstaPLSetFromTransportList
AstaPLSetLongInt
AstaPLSetNull
AstaPLSetSingle
AstaPLSetSmallInt
AstaPLSetString
AstaPLSetTime

AstaParamList

```
Err AstaPLClear(UInt refNum, VoidPtr hPLInstance)
```

Description

Clears the ParamList of all items.

Example

AstaPLClear(reflib,AstaParamList);

AstaParamList

```
Err AstaPLCopyTo(UInt refNum, VoidPtr hPLInstance, VoidPtr hDestPLInstance)
```

Description

Copes one AstaParamList to another.

AstaParamList

```
Err AstaPLCreate(UInt refNum, Int32 InitialSize, VoidPtr *hPLInstance)
```

Description

Creates an AstaParamList

AstaParamList

```
Err AstaPLCreateTransportList(UInt refNum, VoidPtr hPLInstance,
                              VoidPtr *Buffer, Int32 *BufLen)
```

Description
Used internally by AstaLIb.prc to stream an AstaParamList.

AstaParamList

```
Err AstaPLDestroy(UInt refNum, VoidPtr hPLInstance)
```

Description

Destroys an AstaParamList and releases resources.

Example

AstaParamList

```
Err AstaPLGetCount(UInt refNum, VoidPtr hPLInstance, Int32 *Result)
```

Description

Returns the number of items in the AstaParamList

Example
AstaPLGetCount(reflib,ParamList,Count);

AstaParamList

```
Err AstaPLGetParameter(UInt refNum, VoidPtr hPLInstance,
Int32 Index, VoidPtr *hParamInstance)
```

Description

Retrieves an AstaParamValue at Index from an AstaParamList

Example
var
ParamList : pointer;
ParamItem:Pointer;

AstaPlGetParameter(RefLib, ParamList, 0, ParamItem);

AstaParamValue

```
Err AstaPLGetParameterByName(UInt refNum, VoidPtr hPLInstance, CharPtr Name, VoidPtr
*hParamInstance)
```

Description

Retrieves an AstaParamValue by ParamName from an AstaParamList

Example

AstaParamValue

```
Err AstaPLGetParameterIndex(UInt refNum, VoidPtr hPLInstance,VoidPtr hParamInstance, Int32
*Result)
```

Description
Returns the Index of an AstaParamValue from an AstaParamList

AstaParamList

```
Err AstaPLSetBlob(UInt refNum, VoidPtr hPLInstance,
Int32 Index, VoidPtr value, Int32 ValueLen, char *Name)
```

Description

Sets an AstaParamValue with a buffer of Value of Length ValueLen.

Example

AstaPlSetBlob(RefLib,Params,0,TheBlob,BlobSize,'Blob');

AstaParamList

```
Err AstaPLSetBoolean(UInt refNum, VoidPtr hPLInstance,
Int32 Index, bool value, char *Name)
```

Description
Sets an AstaParamValue at Index to to a Boolean value with ParamName Name.

AstaParamList

```
Err AstaPLSetByte(UInt refNum, VoidPtr hPLInstance,
Int32 Index, char value, char *Name)
```

Description

Sets an AstaParamValue at Index to a Byte value with ParamName Name.

Example

AstaParamList

```
Err AstaPLSetDate(UInt refNum, VoidPtr hPLInstance,
Int32 Index, DateTimePtr value, char *Name)
```

Description

Sets an AstaParamValue at Index to a DateType Value with ParamName Name.

Example

AstaParamList

```
Err AstaPLSetDateTime(UInt refNum, VoidPtr hPLInstance,
Int32 Index, DateTimePtr value, char *Name)
```

Description
Sets an AstaParamValue at Index to a DateTime value with ParamName Name.

AstaParamList

```
Err AstaPLSetDouble(UInt refNum, VoidPtr hPLInstance,
Int32 Index, double value, char *Name)
```

Description
Sets a Param Item at Index to to a double value with ParamName Name.

AstaParamList

```
Err AstaPLSetFromTransportList(UInt refNum, VoidPtr hPLInstance,
                               VoidPtr Buf, Int32 BufLen)
```

Description
Used internally to receive a raw buffer and convert to an AstaParamList.

AstaParamList

```
Err AstaPLSetLongInt(UInt refNum, VoidPtr hPLInstance,
Int32 Index, long value, char *Name)
```

Description
Sets an AstaParamValue at Index to LongInt with a ParamName of Name.

Example

```
AstaPlSetLongInt(RefLib,Params,1,45,'Age');
```

AstaParamList

```
Err AstaPLSetNull(UInt refNum, VoidPtr hPLInstance,
Int32 Index, AstaFieldType fType, char *Name)
```

Description
Sets an AstaParamValue to Null.

Example

```
AstaPLSetNull(RefLib,Params,1,pftString,'Department');
```

AstaParamList

```
Err AstaPLSetSingle(UInt refNum, VoidPtr hPLInstance,
Int32 Index, float value, char *Name)
```

Description
Sets a Param Item at Index to to a Single value with ParamName Name.

Example

AstaParamList

```
Err AstaPLSetSmallInt(UInt refNum, VoidPtr hPLInstance,
Int32 Index, Int32 value, char *Name)
```

Description

Sets an AstaParamValue at Index to a SmallInt value with ParamName Name.

Example

AstaParamList

```
Err AstaPLSetString(UInt refNum, VoidPtr hPLInstance,
Int32 Index, CharPtr value, char *Name)
```

Description

Sets a String AstaParamValue at index with ParamName of Name

Example

AstaPlSetString(RefLib, ParamList, 0, 'Select * from Customers', 'PalmSelect');

AstaParamList

```
Err AstaPLSetTime(UInt refNum, VoidPtr hPLInstance,
Int32 Index, DateTimePtr value, char *Name)
```

Description

Sets a Time AstaParamValue at index with ParamName of Name

Example

AstaParamValues are the items contained in the expandable AstaParamLists explained more fully in the ASTA Unified Messaging Initiative

Parameter name - the name of the parameter
Parameter type - the mode of the parameter, in, out, inout, result.
Data type - AstaFieldType which follows the database types (char, bit, integer, etc.)
IsNull flag - a flag indicating if the data is null (a bit representing true or false)

Bytes of data - the actual data as a list of bytes

AstaPLparamGetAsBlob
AstaPLParamGetAsBoolean
AstaPLParamGetAsDate
AstaPLParamGetAsDateTime
AstaPLParamGetAsDouble
AstaPLParam.GetAsLongInt
ASTAPLParamGetAsString
AstaPLParamGetAsStringC
AstaPLParamGetAsTime
AstaPLParamGetName
AstaPLParamGetType
AstaPLParamGetIsNull
AstaPLParamSetName

AstaFieldType=(pftUnknown, pftByte, pftSmallint, pftLongint, pftBoolean, pftSingle, pftDouble, pftDate, pftTime, pftDateTime, pftString, pftBlob);
Description

Value	Meaning
pftUnknown	
pftByte	
pftSmallint	
pftLongint	
pftBoolean	
pftSingle	
pftDouble	
pftDate	
pftTime	
pftDateTime	
pftString	
pftBlob	

AstaParamValue

```
Err AstaPLParamGetAsBlob(UInt refNum, VoidPtr hInstance,
VoidPtr Buffer, Int32 *BufLen)
```

Description
Copies an AstaParamValue Data into Buffer and returns the size of the Blob in BufLen.

AstaParamValue

```
Err AstaPLParamGetAsBoolean(UInt refNum, VoidPtr hInstance, bool *Result)
```

Description

Retrieves the Boolean value of an AstaParamValue in Result:Boolean.

AstaParamValue

```
Err AstaPLParamGetAsDate(UInt refNum, VoidPtr hInstance,
DateTimePtr Date, bool *Succeeded)
```

Description

Retrieves the DateTimePtr value of an AstaParamValue in Date and if successful returns True in Result.

AstaParamValue

```
Err AstaPLParamGetAsDateTime(UInt refNum, VoidPtr hInstance, DateTimePtr Time, bool
*Succeeded)
```

Description

Retrieves the DateTimePtr value of an AstaParamValue in Time and if successful returns True in Result.

AstaParamValue

```
Err AstaPLParamGetAsDouble(UInt refNum, VoidPtr hInstance,
double *Result)
```

Description
Retrieves a double value of an AstaParamValue in Result.

AstaParamValue

```
Err AstaPLParamGetAsLongInt(UInt refNum, VoidPtr Instance, long *Result)
```

Description

Retrieves a LongInt value of an AstaParamValue in Result.

AstaParamValue

```
Err AstaPLParamGetAsString(UInt refNum, VoidPtr hInstance,
CharPtr NameBuf, Int32 *BufLen)
```

Description
Retrieves the AstaParamValue into NameBuf and the length of the Pchar into BufLen.

AstaParamValue

```
Err AstaPLParamGetAsStringC(UInt refNum, VoidPtr hInstance, CharPtr NameBuf)
```

Description
Retrieves the AstaParamValue into NameBuf

AstaParamValue

```
Err AstaPLParamGetAsTime(UInt refNum, VoidPtr hInstance,
DateTimePtr Time, bool *Succeeded)
```

Description

AstaParamValue

```
Err AstaPLParamGetIsNull(UInt refNum, VoidPtr hInstance,
bool *IsNull)
```

Description
Returns True to Result of the if the ParamValue is null.

AstaParamValue

```
Err AstaPLParamGetName(UInt refNum, VoidPtr hInstance,
CharPtr NameBuf, Int32 *BufLen)
```

Description
Retrieves the Name of the ParamValue along with the buffer length.

AstaParamValue

```
Err AstaPLParamGetNameC(UInt refNum, VoidPtr hInstance,
CharPtr NameBuf)
```

Description
Retrieves the Name of the ParamValue.

AstaParamValue

```
Err AstaPLParamGetType(UInt refNum, VoidPtr hInstance,
AstaFieldType *Type)
```

Description

Retrieves a AstaFieldtype value of an AstaParamValue in Result.

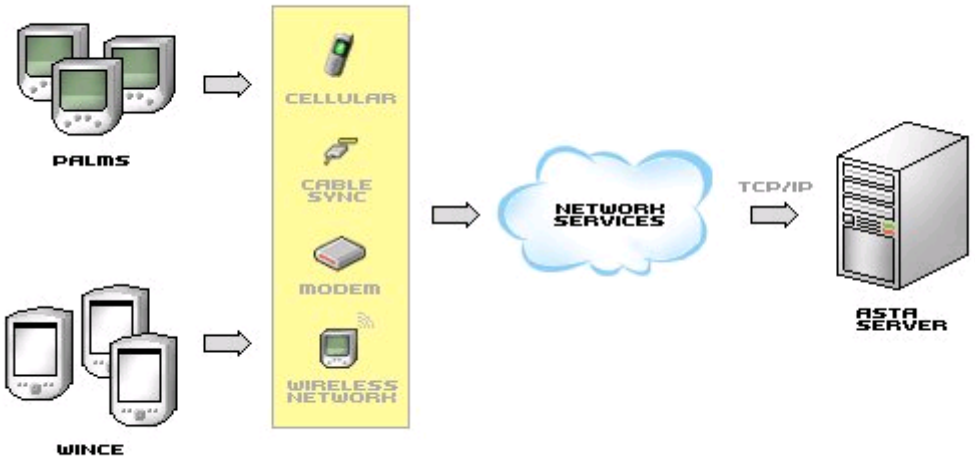
AstaParamValue

```
Err AstaPLParamSetName(UInt refNum, VoidPtr hInstance, CharPtr Value)
```

Description

Sets an AstaParamValue Name to Value

1.10.2.4.3.3 NSBASIC



The ASTA SkyWire architecture consists of remote thin clients, connecting to a middle tier Application Server via TCP/IP. SkyWire Servers are available using a number of languages include Visual Basic, Delphi and Java. This document describes how to use NSBASIC Clients to access remote SkyWire Servers servers and perform standard tasks using the SkyWire component.

ASTA Skywire consists of a middleware Server and client side components to access that server. The following components are used by SkyWire clients to perform standard tasks.

AstaConnection
AstaParamList
AstaParamValue
AstaDataList
AstaNestedParamList

The AstaClientConnection is the client end of the TCP/IP connection. It connects the ASTA client to the ASTA SkyWire Server. The Address and Port properties on the ASTA client must match the Address and Port properties on the ASTA SkyWire Server. Typically the handheld device would have a form where the IPAddress and Port can be input by the user so that the AstaClientConnection can connect to a SkyWire server and authenticate using a Username and Password.

AstaConnClose
AstaConnCreate
AstaConnDestroy
AstaConnGetFile
AstaConnGetLastError

AstaConnGetUserError
AstaConnOpen
AstaConnReceiveParamList
AstaConnSendParamList
AstaConnSetAuthenticationOff
AstaConnSetAuthenticationOn
AstaConnSetCompressionOff
AstaConnSetCompressionOn
AstaConnSetEncryptionOff
AstaConnSetEncryptionOn
AstaConnSetHttpOn
AstaConnSetHttpOff
AstaConnSetTimeout
AstaRequestUpdate

AstaConnection

function AstaConnClose(refNum: UInt16; hConnInstance: Pointer; **var** Result: boolean): Err;

Description

Closes a connection opened after calling AstaConnOpen.

Example

```
AstaConnClose(RefLib, Conversation, BoolRes);
```

AstaConnection

function AstaConnCreate(refNum: UInt16; **var** Result: Pointer): Err;

Description
Creates an AstaConnection as the Var Result:Pointer where refNum represents the handle to the Asta Shared Library.

Example
```
AstaConnCreate(RefLib, Conversation);
```

AstaConnection

function AstaConnDestroy(refNum: UInt16; hConnInstance: pointer): Err;

Description
Destroys and AstaConnection and releases it's resources that has been created with AstaConnOpen.

Example
```
AstaConnDestroy(RefLib, Conversation);
```

AstaConnection

function AstaConnGetFile(refNum :UInt16; hConnInstance :Pointer; BlockSize:UInt32;
RemoteFile:PChar; LocalName :PChar; Var Result:Boolean):Err

Description
Retrieves a remote file from an ASTA Server in BlockSize "chunks". This allows, for example, for large images to be streamed down efficiently.

Example
AstaConnGetFile(refLib, Connection, 8192, RemoteFileName, LocalFileName, Result);

AstaConnection

function AstaConnGetLastError(refNum: UInt16; hConnInstance: Pointer; **var** Result: Int32): Err;

Description
Returns the last error generated by the AstaClientConnection. AstaSky Wire servers are codedso as to return an Integer value on each client request where a non-zero value is considered to be an error.

Example
AstaConnGetLastError(RefLib,Connection,Result)

AstaConnection

function AstaConnGetUserError(refNum: UInt16; hConnInstance: Pointer; **var** Result: Int32): Err;

escription
Returns the last error generated by the AstaClientConnection. AstaSky Wire servers are coded so as to return an Integer value on each client request where a non-zero value is considered to be an error. Typically errors are also returned with the server side error message or exception in an AstaParamValue of Name "Error"

Example
AstaConnGetUserError(RefLib,Connection,Result);

AstaConnection

function AstaConnOpen(refNum: UInt16; hConnInstance: Pointer; Address: PChar; Port: **Word**; **var** Result: boolean): Err;

Description
Attempts to connect to a remote AstaSkyWire server using the Address and Port and will return a False a connection to the server cannot be achieved. No data is transferred to the server. The Asta SkyWire shared library must already be loaded.

Example

```
AstaConnOpen( RefLib, Conversation, '127.0.01',9050, BoolRes);
```

AstaConnection

function AstaConnReceiveParamList(refNum: UInt16; hConnInstance: Pointer; hPLInstance: Pointer; **var** Result: boolean): Err;

Description
Receives an AstaParamList from a remote ASTA Server where the hPLIntance is an AstaParamList that has been created with a call to AstaPLCreate. If the ParamList was sucessfully retrieved from the server the Var Result:Boolean would be set to True.

Example
Typically, if you had an existing Paramlist that you had used in sending params to a remote server you would want to clear it first.

```
AstaPlClear(RefLib, ParamList);
AstaConnReceiveParamList(RefLib, Conversation, ParamList, BoolRes);
```

AstaConnection

function AstaConnSendParamList(refNum: UInt16; hConnInstance: Pointer; hPLInstance: Pointer; MsgToken: UInt32; **var** Result: boolean): Err;

Description
Sends an AstaParam to a remote Server. The AstaParamList must have been created and typically is filled with values before being sent to the remote server. The MsgToken:UInt32 will determine how the server processes the request. If the ParamList was successfully sent to the server, teh Var Result:Boolean will be set to true.

Example
AstaConnSendParamList(RefLib, Conversation, ParamList, 1001, BoolRes);

AstaConnection

function AstaConnSetAuthorizationOff(refNum: UInt16; hConnInstance: Pointer): Err;

Description
Turns off any Authentication information being sent to the AstaSkyWire server on connect.

Example

```
AstaConnSetAuthorizationOff(RefLib,Conversation);
```

AstaConnection

function AstaConnSetAuthorizationOn(refNum: UInt16; hConnInstance: Pointer; UserName: PChar; Password: PChar; SecurePassword: boolean): Err;

Description
Sets the Username and password to be sent to the remote ASTA SkyWire server to authenticate User Information.

Example

```
AstaConnSetAuthorizationOn(RefLib,Conversation,'ASTA','Password',False);
```

AstaConnection

function AstaConnSetCompressionOff(refNum: UInt16; hConnInstance: Pointer): Err;

Description
Sets Compression off. Requires AstaZLib.prc

Example
```
AstaConnSetCompressionOff(RefLib,Conversation);
```

AstaConnection

function AstaConnSetCompressionOn(refNum: UInt16; hConnInstance: Pointer; level: UInt32): Err;

Description
Sets Compression on where level is the compression factor and the higher the number the greater the compresssion but is also slower. Requires AstaZLib.prc

Example
```
AstaConnSetCompressionOn(RefLib,Conversation,9);
```

AstaConnection

function AstaConnSetEncryptionOff(refNum: UInt16; hConnInstance: Pointer): Err;

Description
Sets AstaAES Encryption off.

Example
AstaConnSetEncryptionOff(RefLib,Connection);

AstaConnection

function AstaConnSetEncryptionOn(refNum: UInt16; hConnInstance: Pointer; key: Pointer; keyMode: UInt32): Err;

Description
Sets EncryptionOn and passes in a Key that is used to encrypt and decrypt the data. The Server must be aware of this key. KeyMode signifies which direction and it should usually set to both (2).

Example
AstaConnSetEncryptionOn(`RefLib,Conversation,'MumsTheWord',2);`

AstaConnection

function AstaConnSetHttpOff(refnum: UInt16; hconnInstance: Pointer): Err

Will be available in Astalib 1.1 and necessary for ASTA Palm SOAP support along with the ASTA Palm

XML Parser

Description
Turns off http tunneling that was set with AstaConnSetHttpOn.

AstaConnection

function AstaConnSetHttpOn(refnum: UInt16; hConnInstance: Pointer; Address: PChar; Port: Word;
Page: PChar;
Authenticate: Boolean; UserName, Password: Pchar; UseProxy: Boolean; ProxyAddress: PChar;
ProxyPort: Word; ProxyAuthenticate: Boolean; ProxyUserName: Pchar; ProxyPassWord: Pchar): Err

Will be available in Astalib 1.1 and necessary for ASTA Palm SOAP support along with the ASTA Palm XML Parser

Description
Uses http instead of tcp/ip to allow access through firewalls and through a WebServer using AstaHttp.dll.

Address:Pchar Address of the ASTA Server
Port: Word : Port of the ASTA Server
Authenticate:Boolean': Authenticate at the server
UserName:Pchar UserName
Password: Pchar; Password
UseProxy: Boolean Use a proxy Server
ProxyAddress: PChar; Address of the Proxy Server
ProxyPort: Word Port of the Proxy Server
ProxyAuthenticate: Boolean Autentication required at the Proxy
ProxyUserName: Pchar Username used by Proxy Server
ProxyPassWord: Pchar Password used by Proxy Server

Example

AstaConnection

function AstaConnSetServerVersion(refNum: UInt16; hConnInstance: Pointer; Version: UInt32): Err;

Description
Asta pda libraries can support ASTA for Windows and AstaInterOp for Linux and Windows. This swtich sets the ASTA library.

Valid values are

```
Const
Asta    = 2
AstaIO  = 3;
```

Example

```
AstaConnSetServerVersion(reflib,Connection,Asta);

AstaConnSetServerVersion(reflib,Connection,AstaIO);
```

AstaConnection

function AstaConnSetTimeout(refNum: UInt16; hConnInstance: Pointer; Timeout: UInt32): Err;

Description
Sets the value in Milliseconds that an AstaConnection will "block" in waiting for a response to the server. Since there is no threading on the Palm, sockets must connect to a server and then block or wait.

Example
AstaConnSetTimeout(reflib,Connection,5000);

AstaConnection

function AstaRequestUpdate(refNum: UInt16; hConnInstance: Pointer; Buffer: Pointer; BufLen: UInt32; **var** Result: boolean): Err;

Description

AstaConnection

function AstaSetTerminateConn(refNum: UInt16; hConnInstance: Pointer; value: boolean): Err;

Description

This will the AstaConnection to terminate the connection with the server after each connection. The default is to set this to true and is recommended over any wireless connection. With WinCE clients they may stay connected.

Example

AstaSetTerminateConn(reflib,Connection,True);

The AstaDataList is an efficient DataStructure that handles rows and columns much like a database table. To request a DataList from a remote server you must create an AstaParamList and fill it with the following values:

AstaPLSetString(AstaLibRef, params, 0,'Select * from Customer', "PalmSelect");
AstaPLSetLongInt(AstaLibRef, params, 1, 10000, "RowCount");

Param 0: the Select SQL as a String
Param 1 : the maximum rowcount

The DataList basically contains a List of ParamLists that contain data natively, stored very efficiently, wthout duplicating paramnames, like the AstaNestedParamList does. Each row can be retrieved into a

ParamList with the option of bring the Param or "FieldNames" also by calling AstaDLGetAsParamList

AstaDLCreate
AstaDLDestroy
AstaDLGetAsParamList
AstaDLGetField
AstaDLGetFieldByName
AstaDLGetFieldByNo
AstaDLGetFieldCount
AstaDLGetFieldName
AstaDLGetFieldNameC
AstaDLGetFieldNo
AstaDLGetFieldOptions
AstaDLGetFieldType
AstaDLGetIndexOfField
AstaDLGetRecordsCount
AstaDLIsEmpty

AstaDataList

function AstaDLCreate(refNum: UInt16; Buffer: Pointer; BufLen: UInt32; **var** Result: Pointer): Err;

Description
A DataList is created on a remote ASTA server in response to a select SQL . PdaMultiSQLSelect, or custom result set being streamed back to an ASTA Pda client.

AstaDLCreate Creates a DataList from a paramList AstaParamValue streamed over the wire as a Blob Param.

Example
This examples assumes the ParamList has been received using AstaConnReciveParamList. The ReturnParam:Pointer below is an AstaParamValue.

```
AstaPLGetParameter(refLib, ParamList, 0, ReturnParam);

DataPtr :=nil;
DataLen := 0;
//with Dataptr as NIL the size of the blob is returned in DataLen
AstaPLParamGetAsBlob(RefLib, ReturnParam, DataPtr, DataLen);

DataPtr := MemPtrNew(DataLen+1);//allocate some memory now

AstaPLParamGetAsBlob(Reflib, ReturnParam, DataPtr, DataLen);

DataList:=nil;

AstaDLCreate(Reflib, DataPtr, DataLen, DataList);

MemPtrFree(DataPtr);
```

AstaDataList

function AstaDLDestroy(refNum: UInt16; hDLInstance: Pointer): Err;

Description
Destroys an AstaDataList and frees up it's resources;

Example
AstaDLDestroy(refLib,DataList);

AstaDataList

function AstaDLGetAsParamList(refNum: UInt16; hDLInstance: Pointer; recordNumber: UInt32; putNames: boolean; **var** Result: Pointer): Err;

Description
Retrieves an AstaParamList at the row designated as RecordNumber into the Var Result:Pointer. The number of rows or records can be obtained by calling AstaDLGetRecordsCount.

Example
AstaDLGetAsParamList(refLib,DataList,1,True,TempParamList);

AstaDataList

function AstaDLGetField(refNum: UInt16; hDLInstance: Pointer; **Index**: UInt32; **var** Result: Pointer): Err;

Description
Returns a handle to a Field by a given Index.

AstaDataList

function AstaDLGetFieldByName(refNum: UInt16; hDLInstance: Pointer; **Name**: PChar; **var** Result: Pointer): Err;

Description
Returns the "handle" to the "current" field by given name

AstaDataList

function AstaDLGetFieldByNo(refNum: UInt16; hDLInstance: Pointer; **Index**: UInt32; **var** Result: Pointer): Err;

Description

Returns the "handle" to the field by given index

AstaDataList

function AstaDLGetFieldCount(refNum: UInt16; hDLInstance: Pointer; **var** Result: UInt32): Err;

Description
Returns the number of fields in a DataLis in the Var Result: UInt32.

Example
AstaDLGetFieldCount(refLib,DataList,Fieldcount);

AstaDataList

function AstaDLGetFieldName(refNum: UInt16; hDLInstance: Pointer; hFieldInstance: Pointer; NameBuf: PChar; **var** BufLen: UInt32): Err;

Description
Returns the fieldname of a given field when a Field Handle is passed in as well as the length of the field in BufLen

AstaDataList

function AstaDLGetFieldNameC(refNum: UInt16; hDLInstance: Pointer; hFieldInstance: Pointer; NameBuf: PChar): Err;

Description
Returns the name of the field

AstaDataList

function AstaDLGetFieldNo(refNum: UInt16; hDLInstance: Pointer; hFieldInstance: Pointer; **var** Result: UInt32): Err;

Description
Returns the Nnumber of the field.

AstaDataList

function AstaDLGetFieldOptions(refNum: UInt16; hDLInstance: Pointer; hFieldInstance: Pointer; **var** Result: UInt32): Err;

```
#define pfaHiddenCol    1
#define pfaReadonly     2
#define pfaRequired     4
```

```
#define pfaLink          8
#define pfaUnNamed       16
#define pfaFixed                 3
```

Description
Returns the options of the field.

AstaDataList

function AstaDLGetFieldType(refNum: UInt16; hDLInstance: Pointer; hFieldInstance: Pointer; **var** Result: AstaFieldType): Err;

Description
Returns the type of the field for a given Field Handle

AstaDataList

function AstaDLGetIndexOfField(refNum: UInt16; hDLInstance: Pointer; hFieldInstance: Pointer; **var** Result: UInt32): Err;

Description
Returns the "index" of the field in Result of a given Field Handle.

AstaDataList

function AstaDLGetRecordsCount(refNum: UInt16; hDLInstance: Pointer; **var** Result: UInt32): Err;

Description
Returns the number of records contained in an AstaDataList in Result. This count usually corresponds to the number of rows requested to be returned from a remote server.

Example

```
AstaDLGetRecordsCount(reflib,DataList,RecordCount);
```

AstaDataList

function AstaDLIsEmpty(refNum: UInt16; hDLInstance: Pointer; **var** Result: boolean): Err;

Description

If Result then the AstaDataList is Empty.

The AstaNestedParamList provides a datatructure that works like a DataSet with a row/column abstraction in order to handle SQL result requests from ASTA SkyWire servers with all the data stored as Strings. This makes it easy to use with User Interface Controls. To handle data natively, and a bit more efficiently use DataLists. NestedParamLists are actually an AstaParamList of AstaParamLists with the first ParamList a list of Fields and the rest of the paramlists containing the actual SQL Result sets.

Nested Param List Example

AstaNPLCreate
ASTANPLDestroy
AstaNPLGetField
AstaNPLGetFieldByName
AstaNPLGetFieldCount
AstaNPLGetRecordsCount
AstaNPLIsBOF
AstaNPLisEmpty
AstaNPLIsEOF
AstaNPLNext
AstaNPLPrevious

AstaNestedParamList

function AstaNPLCreate(refNum: UInt16; hPLInstance: Pointer; **var** Result: Pointer): Err;

Description
Creates an AstaNestedParamList in Result from an incoming ParamList from the server in response to PdaToken PdaNestedSQLSelect (1010)

Example

```
AstaNPLCreate(RefLib, ParamList, NPL);
```

AstaNestedParamList

function AstaNPLDestroy(refNum: UInt16; hPLInstance: Pointer): Err;

Description
Disposes of an AstaNestedfParamList previously created with AstaNPLCreate and releases it's resources.

Example

```
AstaNPLDestroy(RefLib, NPL);
```

AstaNestedParamList

function AstaNPLGetField(refNum: UInt16; hPLInstance: Pointer; **Index**: UInt32; **var** Result: Pointer): Err;

Description
Retrieves a Field as an AstaParamValue from a NestedParamList.

Example

```
for i := 0 to ParamCount - 1 do
    begin
  Value := nil;
  ValueLen := 0;
  AstaNPLGetField(RefLib, NPL, i, Fld);
  if (fld <> nil) then
  begin
    AstaPLParamGetAsString(RefLib, fld, Value, ValueLen);
    Inc(ValueLen);
    Value := MemPtrNew(ValueLen);
    AstaPLParamGetAsString(RefLib, fld, Value, ValueLen);
    Params[i] := Value;
  end;
 end;
```

AstaNestedParamList

function AstaNPLGetFieldByName(refNum: UInt16; hPLInstance: Pointer; **Name**: PChar; **var** Result: Pointer): Err;

Description

Retrieves an AstaNestedParamList Field byName into Result.

Example

AstaNestedParamList

function AstaNPLGetFieldCount(refNum: UInt16; hPLInstance: Pointer; **var** Result: UInt32): Err;

Description
Returns the number of Fields in a AstaNestedParamList

Example
AstaNPLGetFieldCount(RefLib, NPL, ParamCount);

AstaNestedParamList

function AstaNPLGetRecordsCount(refNum: UInt16; hPLInstance: Pointer; **var** Result: **Integer**): Err;

Description

Returns the number of rows in an AstaNestedParamList where each row is an AstaParamList

Example
AstaNPLGetRecordsCount(RefLib, NPL, Count);

AstaNestedParamList

function AstaNPLIsBOF(refNum: UInt16; hPLInstance: Pointer; **var** Result: boolean): Err;

Description

Returns whether the postion of the AstaNestedParamList is Beginning of File similar to the TDataSet.BOF Call

AstaNestedParamList

function AstaNPLIsEmpty(refNum: UInt16; hPLInstance: Pointer; **var** Result: boolean): Err;

Description
Checks to see if an AstaNestedParamList is Empty and returns a boolean in Var Result if it is.

Example

AstaNPLIsEmpty(RefLib, NPL, BoolRes);

AstaNestedParamList

function AstaNPLIsEOF(refNum: UInt16; hPLInstance: Pointer; **var** Result: boolean): Err;

Description

Returns whether the postion of the AstaNestedParamList is End of File similar to the TDataSet.EOF Call

Example

AstaNestedParamList

function AstaNPLNext(refNum: UInt16; hPLInstance: Pointer; **var** Result: boolean): Err;

Description

Moves to the next embedded AstaParamList from an AstaNestedParamlist

AstaNestedParamList

function AstaNPLPrevious(refNum: UInt16; hPLInstance: Pointer; **var** Result: boolean): Err;

Description

Moves to the previous embedded AstaParamList from an AstaNestedParamlist

The ASTA Unified Messaging Initiative allows for cross platform messaging by using AstaParamLists which are easy to use and fully streamable Data containers that can be transported between ASTA clients and servers and between ASTA clients regardless of hardware or operating system.

AstaPLDestroy
AstaPLClear
AstaPLCopyTo
AstaPLCreate
AstaPLCreateTransportList
AstaPLGetCount
AstaPLGetParameter
AstaPLGetParameterByName
AstaPLGetParameterIndex
AstaPLSetBlob
AstaPLSetBoolean
AstaPLSetByte
AstaPLSetDate
AstaPLSetDateTime
AstaPLSetDouble

AstaPLSetFromTransportList
AstaPLSetLongInt
AstaPLSetNull
AstaPLSetSingle
AstaPLSetSmallInt
AstaPLSetString
AstaPLSetTime

AstaParamList

function AstaPLClear(refNum: UInt16; hPLInstance: Pointer): Err;

Description

Clears the ParamList of all items.

Example

AstaPLClear(reflib,AstaParamList);

AstaParamList

function AstaPLCopyTo(refNum: UInt16; hPLInstance: Pointer; hDestPLInstance: Pointer): Err;

Description

Copes one AstaParamList to another.

AstaParamList

function AstaPLCreate(refNum: UInt16; InitialSize: Int32; **var** Result: Pointer): Err;

Description

Creates an AstaParamList

Example
This creates an AstaParamlist with 5 initial ParamItems.

```
var
 ParamList:Pointer;

AstaPLCreate(reflib,5,ParamList);
```

AstaParamList

function AstaPLCreateTransportList(refNum: UInt16; hPLInstance: Pointer; **var** Buffer: Pointer; **var** BufLen: **integer**): Err;

Description
Used internally by AstaLIb.prc to stream an AstaParamList.

AstaParamList

function AstaPLDestroy(refNum: UInt16; hPLInstance: Pointer): Err;

Description

Destroys an AstaParamList and releases resources.

Example

AstaParamList

function AstaPLGetCount(refNum: UInt16; hPLInstance: Pointer; **var** Result: UInt32): Err;

Description

Returns the number of items in the AstaParamList

Example
AstaPLGetCount(reflib,ParamList,Count);

AstaParamList

function AstaPLGetParameter(refNum: UInt16; hPLInstance: Pointer; **Index**: UInt32; **var** Result: Pointer): Err;

Description

Retrieves an AstaParamValue at Index from an AstaParamList

Example
var
ParamList : pointer;
ParamItem:Pointer;

AstaPlGetParameter(RefLib, ParamList, 0, ParamItem);

AstaParamList

function AstaPLSetBlob(refNum: UInt16; hPLInstance: Pointer; **Index**: UInt32; value: Pointer; ValueLen: UInt32; **Name**:

PChar): Err;

Description

Sets an AstaParamValue with a buffer of Value of Length ValueLen.

Example

AstaPlSetBlob(RefLib,Params,0,TheBlob,BlobSize,'Blob');

AstaParamList

function AstaPLSetBoolean(refNum: UInt16; hPLInstance: Pointer; **Index**: UInt32; value: boolean; **Name**: PChar): Err;

Description

Sets an AstaParamValue at Index to to a Boolean value with ParamName Name.

AstaParamList

function AstaPLSetByte(refNum: UInt16; hPLInstance: Pointer; **Index**: UInt32; Value: char; **Name**: PChar): Err;

Description

Sets an AstaParamValue at Index to a Byte value with ParamName Name.

Example

AstaParamList

function AstaPLSetDate(refNum: UInt16; hPLInstance: Pointer; **Index**: UInt32; Value: DateTimePtr; **Name**: PChar): Err;

Description

Sets an AstaParamValue at Index to a DateType Value with ParamName Name.

Example

AstaParamList

function AstaPLSetDateTime(refNum: UInt16; hPLInstance: Pointer; **Index**: UInt32; Value: DateTimePtr; **Name**: PChar): Err;

Description

Sets an AstaParamValue at Index to a DateTime value with ParamName Name.

AstaParamList

function AstaPLSetDouble(refNum: UInt16; hPLInstance: Pointer; **Index**: UInt32; value: double; **Name**: PChar): Err;

Description

Sets a Param Item at Index to to a double value with ParamName Name.

Example
AstaPlSetDouble(RefLib,Params,1,5.44,'Double');

AstaParamList

function AstaPLSetFromTransportList(refNum: UInt16; hPLInstance: Pointer; Buf: Pointer; BufLen: UInt32): Err;

Description
Used internally to receive a raw buffer and convert to an AstaParamList.

AstaParamList

function AstaPLSetLongInt(refNum: UInt16; hPLInstance: Pointer; **Index**: UInt32; value: Int32; **Name**: PChar): Err;

Description
Sets an AstaParamValue at Index to LongInt with a ParamName of Name.

Example

AstaPlSetLongInt(RefLib,Params,1,45,'Age');

AstaParamList

function AstaPLSetNull(refNum: UInt16; hPLInstance: Pointer; **Index**: UInt32; fType: AstaFieldType; **Name**: PChar): Err;

Description

Sets an AstaParamValue to Null.

Example
AstaPLSetNull(RefLib,Params,1,pftString,'Department');

AstaParamList

function AstaPLSetSingle(refNum: UInt16; hPLInstance: Pointer; **Index**: UInt32; value: single; **Name**: PChar): Err;

Description
Sets a Param Item at Index to to a Single value with ParamName Name.

Example

AstaParamList

function AstaPLSetSmallInt(refNum: UInt16; hPLInstance: Pointer; **Index**: UInt32; value: Int32; **Name**: PChar): Err;

Description

Sets an AstaParamValue at Index to a SmallInt value with ParamName Name.

Example

AstaParamList

function AstaPLSetString(refNum: UInt16; hPLInstance: Pointer; **Index**: UInt32; Value: PChar; **Name**: PChar): Err;

Description

Sets a String AstaParamValue at index with ParamName of Name

Example

AstaPlSetString(RefLib, ParamList, 0, 'Select * from Customers', 'PalmSelect');

AstaParamList

function AstaPLSetTime(refNum: UInt16; hPLInstance: Pointer; **Index**: UInt32; Value: DateTimePtr; **Name**: PChar): Err;

Description

Sets a Time AstaParamValue at index with ParamName of Name

Example

AstaParamValues are the items contained in the expandable AstaParamLists explained more fully in the ASTA Unified Messaging Initiative

Parameter name - the name of the parameter
Parameter type - the mode of the parameter, in, out, inout, result.
Data type - AstaFieldType which follows the database types (char, bit, integer, etc.)
IsNull flag - a flag indicating if the data is null (a bit representing true or false)
Bytes of data - the actual data as a list of bytes

AstaPLparamGetAsBlob
AstaPLParamGetAsBoolean
AstaPLParamGetAsDate
AstaPLParamGetAsDateTime
AstaPLParamGetAsDouble
AstaPLParam.GetAsLongInt
ASTAPLParamGetAsString
AstaPLParamGetAsStringC
AstaPLParamGetAsTime
AstaPLParamGetName
AstaPLParamGetType
AstaPLParamGetIsNull
AstaPLParamSetName

AstaFieldType=(pftUnknown, pftByte, pftSmallint, pftLongint, pftBoolean, pftSingle, pftDouble, pftDate, pftTime, pftDateTime, pftString, pftBlob);
Description

Value	**Meaning**
pftUnknown	
pftByte	
pftSmallint	
pftLongint	
pftBoolean	
pftSingle	
pftDouble	
pftDate	
pftTime	
pftDateTime	
pftString	
pftBlob	

AstaParamValue

function AstaPLGetParameterByName(refNum: UInt16; hPLInstance: Pointer; **Name**: PChar; **var** Result: Pointer): Err;

Description

Retrieves an AstaParamValue by ParamName from an AstaParamList

Example

AstaParamValue

function AstaPLGetParameterIndex(refNum: UInt16; hPLInstance: Pointer; hParamInstance: Pointer; **var** Result: UInt32): Err;

Description

Returns the Index of an AstaParamValue from an AstaParamList

Example

AstaParamValue

function AstaPLParamGetAsBlob(refNum: UInt16; hInstance: Pointer; Buffer: Pointer; **var** BufLen: UInt32): Err;

Description

Copies an AstaParamValue Data into Buffer and returns the size of the Blob in BufLen.

AstaParamValue

function AstaPLParamGetAsBoolean(refNum: UInt16; hInstance: Pointer; **var** Result: boolean): Err;

Description

Retrieves the Boolean value of an AstaParamValue in Result:Boolean.

AstaParamValue

function AstaPLParamGetAsDate(refNum: UInt16; hInstance: Pointer; Date: DateTimePtr; **var** Result: boolean): Err;

Description

Retrieves the DateTimePtr value of an AstaParamValue in Date and if successful returns True in Result.

AstaParamValue

function AstaPLParamGetAsDateTime(refNum: UInt16; hInstance: Pointer; Time: DateTimePtr; **var**

Result: boolean): Err;

Description

Retrieves the DateTimePtr value of an AstaParamValue in Time and if successful returns True in Result.

AstaParamValue

function AstaPLParamGetAsDouble(refNum: UInt16; hInstance: Pointer; **var** Result: double): Err;

Description

Retrieves a double value of an AstaParamValue in Result.

AstaParamValue

function AstaPLParamGetAsLongInt(refNum: UInt16; hInstance: Pointer; **var** Result: Int32): Err;

Description

Retrieves a LongInt value of an AstaParamValue in Result.

AstaParamValue

function AstaPLParamGetAsString(refNum: UInt16; hInstance: Pointer; NameBuf: PChar; **var** BufLen: UInt32): Err;

Description
Retrieves the AstaParamValue into NameBuf and the length of the Pchar into BufLen.

AstaParamValue

function AstaPLParamGetAsStringC(refNum: UInt16; hInstance: Pointer; NameBuf: PChar): Err;

Description
Retrieves the AstaParamValue into NameBuf

AstaParamValue

function AstaPLParamGetAsTime(refNum: UInt16; hInstance: Pointer; Time: DateTimePtr; **var** Result: boolean): Err;

Description

AstaParamValue

function AstaPLParamGetIsNull(refNum: UInt16; hInstance: Pointer; **var** Result: boolean): Err;

Description
Returns True to Result of the if the ParamValue is null.

AstaParamValue

function AstaPLParamGetName(refNum: UInt16; hInstance: Pointer; NameBuf: PChar; **var** BufLen: UInt32): Err;

Description
Retrieves the Name of the ParamValue along with the buffer length.

AstaParamValue

function AstaPLParamGetNameC(refNum: UInt16; hInstance: Pointer; NameBuf: PChar): Err;

Description
Retrieves the Name of the ParamValue.

Example

AstaParamValue

function AstaPLParamGetType(refNum: UInt16; hInstance: Pointer; **var** Result: AstaFieldType): Err;

Description

Retrieves a AstaFieldtype value of an AstaParamValue in Result.

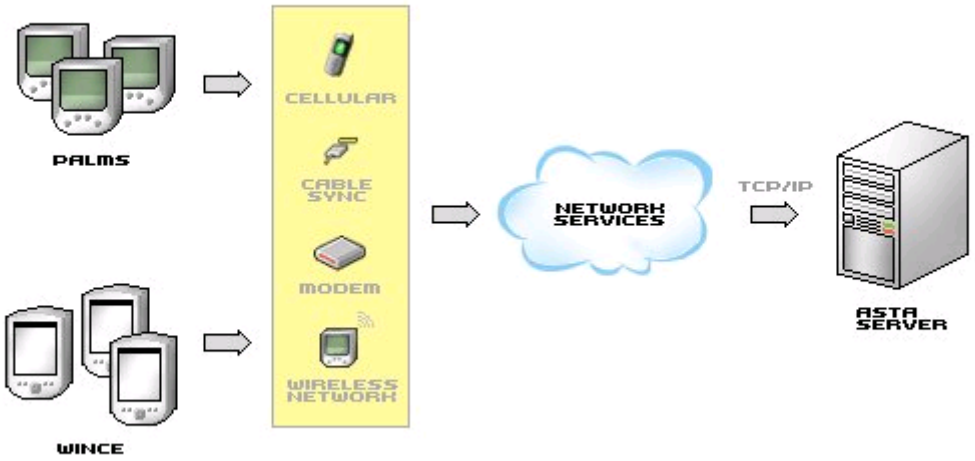
AstaParamValue

function AstaPLParamSetName(refNum: UInt16; hInstance: Pointer; Value: PChar): Err;

Description

Sets an AstaParamValue Name to Value

1.10.2.4.3.4 PocketStudio



The ASTA SkyWire architecture consists of remote thin clients, connecting to a middle tier Application Server via TCP/IP. SkyWire Servers are available using a number of languages include Visual Basic, Delphi and Java. This document describes how to use PocketStudio Clients to access remote SkyWire Servers servers and perform standard tasks using the SkyWire component.

ASTA Skywire consists of a middleware Server and client side components to access that server. The following components are used by SkyWire clients to perform standard tasks.

AstaConnection
AstaParamList
AstaParamValue
AstaDataList
AstaNestedParamList

The AstaClientConnection is the client end of the TCP/IP connection. It connects the ASTA client to the ASTA SkyWire Server. The Address and Port properties on the ASTA client must match the Address and Port properties on the ASTA SkyWire Server. Typically the handheld device would have a form where the IPAddress and Port can be input by the user so that the AstaClientConnection can connect to a SkyWire server and authenticate using a Username and Password.

AstaConnClose
AstaConnCreate
AstaConnDestroy
AstaConnGetFile
AstaConnGetLastError
AstaConnGetUserError
AstaConnOpen
AstaConnReceiveParamList
AstaConnSendParamList
AstaConnSetAuthenticationOff
AstaConnSetAuthenticationOn
AstaConnSetCompressionOff
AstaConnSetCompressionOn
AstaConnSetEncryptionOff
AstaConnSetEncryptionOn
AstaConnSetHttpOn
AstaConnSetHttpOff
AstaConnSetTimeout
AstaRequestUpdate

AstaConnection

function AstaConnClose(refNum: UInt16; hConnInstance: Pointer; **var** Result: boolean): Err;

Description

Closes a connection opened after calling AstaConnOpen.

Example

```
AstaConnClose(RefLib, Conversation, BoolRes);
```

AstaConnection

function AstaConnCreate(refNum: UInt16; **var** Result: Pointer): Err;

Description
Creates an AstaConnection as the Var Result:Pointer where refNum represents the handle to the Asta Shared Library.

Example
```
AstaConnCreate(RefLib, Conversation);
```

AstaConnection

function AstaConnDestroy(refNum: UInt16; hConnInstance: pointer): Err;

Description
Destroys and AstaConnection and releases it's resources that has been created with AstaConnOpen.

Example
```
AstaConnDestroy(RefLib, Conversation);
```

AstaConnection

function AstaConnGetFile(refNum :UInt16; hConnInstance :Pointer; BlockSize:UInt32;
RemoteFile:PChar; LocalName :PChar; Var Result:Boolean):Err

Description
Retrieves a remote file from an ASTA Server in BlockSize "chunks". This allows, for example, for large images to be streamed down efficiently.

Example
AstaConnGetFile(refLib, Connection, 8192, RemoteFileName, LocalFileName, Result);

AstaConnection

function AstaConnGetLastError(refNum: UInt16; hConnInstance: Pointer; **var** Result: Int32): Err;

Description
Returns the last error generated by the AstaClientConnection. AstaSky Wire servers are codedso as to return an Integer value on each client request where a non-zero value is considered to be an error.

Example
AstaConnGetLastError(RefLib,Connection,Result)

AstaConnection

function AstaConnGetUserError(refNum: UInt16; hConnInstance: Pointer; **var** Result: Int32): Err;

escription
Returns the last error generated by the AstaClientConnection. AstaSky Wire servers are coded so as to return an Integer value on each client request where a non-zero value is considered to be an error. Typically errors are also returned with the server side error message or exception in an AstaParamValue of Name "Error"

Example
AstaConnGetUserError(RefLib,Connection,Result);

AstaConnection

function AstaConnOpen(refNum: UInt16; hConnInstance: Pointer; Address: PChar; Port: **Word**; **var** Result: boolean): Err;

Description
Attempts to connect to a remote AstaSkyWire server using the Address and Port and will return a False a connection to the server cannot be achieved. No data is transferred to the server. The Asta SkyWire shared library must already be loaded.

Example
```
AstaConnOpen( RefLib, Conversation, '127.0.01',9050, BoolRes);
```

AstaConnection

function AstaConnReceiveParamList(refNum: UInt16; hConnInstance: Pointer; hPLInstance: Pointer; **var** Result: boolean): Err;

Description
Receives an AstaParamList from a remote ASTA Server where the hPLIntance is an AstaParamList that has been created with a call to AstaPLCreate. If the ParamList was sucessfully retrieved from the server the Var Result:Boolean would be set to True.

Example
Typically, if you had an existing Paramlist that you had used in sending params to a remote server you would want to clear it first.

```
AstaPlClear(RefLib, ParamList);
AstaConnReceiveParamList(RefLib, Conversation, ParamList, BoolRes);
```

AstaConnection

function AstaConnSendParamList(refNum: UInt16; hConnInstance: Pointer; hPLInstance: Pointer; MsgToken: UInt32; **var** Result: boolean): Err;

Description
Sends an AstaParam to a remote Server. The AstaParamList must have been created and typically is filled with values before being sent to the remote server. The MsgToken:UInt32 will determine how the server processes the request. If the ParamList was successfully sent to the server, teh Var Result:Boolean will be set to true.

Example
AstaConnSendParamList(RefLib, Conversation, ParamList, 1001, BoolRes);

AstaConnection

function AstaConnSetAuthorizationOff(refNum: UInt16; hConnInstance: Pointer): Err;

Description
Turns off any Authentication information being sent to the AstaSkyWire server on connect.

Example
```
AstaConnSetAuthorizationOff(RefLib,Conversation);
```

AstaConnection

function AstaConnSetAuthorizationOn(refNum: UInt16; hConnInstance: Pointer; UserName: PChar; Password: PChar; SecurePassword: boolean): Err;

Description
Sets the Username and password to be sent to the remote ASTA SkyWire server to authenticate User Information.

Example
```
AstaConnSetAuthorizationOn(RefLib,Conversation,'ASTA','Password',False);
```

AstaConnection

function AstaConnSetCompressionOff(refNum: UInt16; hConnInstance: Pointer): Err;

Description
Sets Compression off. Requires AstaZLib.prc

Example
```
AstaConnSetCompressionOff(RefLib,Conversation);
```

AstaConnection

function AstaConnSetCompressionOn(refNum: UInt16; hConnInstance: Pointer; level: UInt32): Err;

Description
Sets Compression on where level is the compression factor and the higher the number the greater the compresssion but is also slower. Requires AstaZLib.prc

Example
```
AstaConnSetCompressionOn(RefLib,Conversation,9);
```

AstaConnection

function AstaConnSetEncryptionOff(refNum: UInt16; hConnInstance: Pointer): Err;

Description
Sets AstaAES Encryption off.

Example
AstaConnSetEncryptionOff(RefLib,Connection);

AstaConnection

function AstaConnSetEncryptionOn(refNum: UInt16; hConnInstance: Pointer; key: Pointer; keyMode: UInt32): Err;

Description
Sets EncryptionOn and passes in a Key that is used to encrypt and decrypt the data. The Server must be aware of this key. KeyMode signifies which direction and it should usually set to both (2).

Example
AstaConnSetEncryptionOn(`RefLib,Conversation,'MumsTheWord',2);`

AstaConnection

function AstaConnSetHttpOff(refnum: UInt16; hconnInstance: Pointer): Err

Will be available in Astalib 1.1 and necessary for ASTA Palm SOAP support along with the ASTA Palm XML Parser

Description
Turns off http tunneling that was set with AstaConnSetHttpOn.

AstaConnection

function AstaConnSetHttpOn(refnum: UInt16; hConnInstance: Pointer; Address: PChar; Port: Word; Page: PChar;
Authenticate: Boolean; UserName, Password: Pchar; UseProxy: Boolean; ProxyAddress: PChar;
ProxyPort: Word; ProxyAuthenticate: Boolean; ProxyUserName: Pchar; ProxyPassWord: Pchar): Err

Will be available in Astalib 1.1 and necessary for ASTA Palm SOAP support along with the ASTA Palm XML Parser

Description
Uses http instead of tcp/ip to allow access through firewalls and through a WebServer using AstaHttp.dll.

Address:Pchar Address of the ASTA Server
Port: Word : Port of the ASTA Server
Authenticate:Boolean': Authenticate at the server

UserName:Pchar UserName
Password: Pchar; Password
UseProxy: Boolean Use a proxy Server
ProxyAddress: PChar; Address of the Proxy Server
ProxyPort: Word Port of the Proxy Server
ProxyAuthenticate: Boolean Autentication required at the Proxy
ProxyUserName: Pchar Username used by Proxy Server
ProxyPassWord: Pchar Password used by Proxy Server

Example

AstaConnection

function AstaConnSetServerVersion(refNum: UInt16; hConnInstance: Pointer; Version: UInt32): Err;

Description
Asta pda libraries can support ASTA for Windows and AstaInterOp for Linux and Windows. This swtich sets the ASTA library.

Valid values are

```
Const
Asta    = 2
AstaIO  = 3;
```

Example

```
AstaConnSetServerVersion(reflib,Connection,Asta);

AstaConnSetServerVersion(reflib,Connection,AstaIO);
```

AstaConnection

function AstaConnSetTimeout(refNum: UInt16; hConnInstance: Pointer; Timeout: UInt32): Err;

Description
Sets the value in Milliseconds that an AstaConnection will "block" in waiting for a response to the server. Since there is no threading on the Palm, sockets must connect to a server and then block or wait.

Example

```
AstaConnSetTimeout(reflib,Connection,5000);
```

AstaConnection

function AstaRequestUpdate(refNum: UInt16; hConnInstance: Pointer; Buffer: Pointer; BufLen: UInt32; **var** Result: boolean): Err;

Description

AstaConnection

function AstaSetTerminateConn(refNum: UInt16; hConnInstance: Pointer; value: boolean): Err;

Description

This will the AstaConnection to terminate the connection with the server after each connection. The default is to set this to true and is recommended over any wireless connection. With WinCE clients they may stay connected.

Example

AstaSetTerminateConn(reflib,Connection,True);

The AstaDataList is an efficient DataStructure that handles rows and columns much like a database table. To request a DataList from a remote server you must create an AstaParamList and fill it with the following values:

AstaPLSetString(AstaLibRef, params, 0,'Select * from Customer', "PalmSelect");
AstaPLSetLongInt(AstaLibRef, params, 1, 10000, "RowCount");

Param 0: the Select SQL as a String
Param 1 : the maximum rowcount

The DataList basically contains a List of ParamLists that contain data natively, stored very efficiently, wthout duplicating paramnames, like the AstaNestedParamList does. Each row can be retrieved into a ParamList with the option of bring the Param or "FieldNames" also by calling AstaDLGetAsParamList

AstaDLCreate
AstaDLDestroy
AstaDLGetAsParamList
AstaDLGetField
AstaDLGetFieldByName
AstaDLGetFieldByNo
AstaDLGetFieldCount
AstaDLGetFieldName
AstaDLGetFieldNameC
AstaDLGetFieldNo
AstaDLGetFieldOptions
AstaDLGetFieldType
AstaDLGetIndexOfField
AstaDLGetRecordsCount
AstaDLIsEmpty

AstaDataList

function AstaDLCreate(refNum: UInt16; Buffer: Pointer; BufLen: UInt32; **var** Result: Pointer): Err;

Description
A DataList is created on a remote ASTA server in response to a select SQL . PdaMultiSQLSelect, or custom result set being streamed back to an ASTA Pda client.

AstaDLCreate Creates a DataList from a paramList AstaParamValue streamed over the wire as a Blob Param.

Example
This examples assumes the ParamList has been received using AstaConnReciveParamList. The ReturnParam:Pointer below is an AstaParamValue.

```
AstaPLGetParameter(refLib, ParamList, 0, ReturnParam);

DataPtr :=nil;
DataLen := 0;
//with Dataptr as NIL the size of the blob is returned in DataLen
AstaPLParamGetAsBlob(RefLib, ReturnParam, DataPtr, DataLen);

DataPtr := MemPtrNew(DataLen+1);//allocate some memory now

AstaPLParamGetAsBlob(Reflib, ReturnParam, DataPtr, DataLen);

DataList:=nil;

AstaDLCreate(Reflib, DataPtr, DataLen, DataList);

MemPtrFree(DataPtr);
```

AstaDataList

function AstaDLDestroy(refNum: UInt16; hDLInstance: Pointer): Err;

Description
Destroys an AstaDataList and frees up it's resources;

Example
AstaDLDestroy(refLib,DataList);

AstaDataList

function AstaDLGetAsParamList(refNum: UInt16; hDLInstance: Pointer; recordNumber: UInt32; putNames: boolean; **var** Result: Pointer): Err;

Description
Retrieves an AstaParamList at the row designated as RecordNumber into the Var Result:Pointer. The number of rows or records can be obtained by calling AstaDLGetRecordsCount.

Example
```
AstaDLGetAsParamList(refLib,DataList,1,True,TempParamList);
```

AstaDataList

function AstaDLGetField(refNum: UInt16; hDLInstance: Pointer; **Index**: UInt32; **var** Result: Pointer): Err;

Description
Returns a handle to a Field by a given Index.

AstaDataList

function AstaDLGetFieldByName(refNum: UInt16; hDLInstance: Pointer; **Name**: PChar; **var** Result: Pointer): Err;

Description
Returns the "handle" to the "current" field by given name

AstaDataList

function AstaDLGetFieldByNo(refNum: UInt16; hDLInstance: Pointer; **Index**: UInt32; **var** Result: Pointer): Err;

Description
Returns the "handle" to the field by given index

AstaDataList

function AstaDLGetFieldCount(refNum: UInt16; hDLInstance: Pointer; **var** Result: UInt32): Err;

Description
Returns the number of fields in a DataLis in the Var Result: UInt32.

Example
AstaDLGetFieldCount(refLib,DataList,Fieldcount);

AstaDataList

function AstaDLGetFieldName(refNum: UInt16; hDLInstance: Pointer; hFieldInstance: Pointer; NameBuf: PChar; **var** BufLen: UInt32): Err;

Description
Returns the fieldname of a given field when a Field Handle is passed in as well as the length of the field in BufLen

AstaDataList

function AstaDLGetFieldNameC(refNum: UInt16; hDLInstance: Pointer; hFieldInstance: Pointer; NameBuf: PChar): Err;

Description
Returns the name of the field

AstaDataList

function AstaDLGetFieldNo(refNum: UInt16; hDLInstance: Pointer; hFieldInstance: Pointer; **var** Result: UInt32): Err;

Description
Returns the Nnumber of the field.

AstaDataList

function AstaDLGetFieldOptions(refNum: UInt16; hDLInstance: Pointer; hFieldInstance: Pointer; **var** Result: UInt32): Err;

```
#define pfaHiddenCol    1
#define pfaReadonly     2
#define pfaRequired     4
#define pfaLink         8
#define pfaUnNamed      16
#define pfaFixed            3
```

Description
Returns the options of the field.

AstaDataList

function AstaDLGetFieldType(refNum: UInt16; hDLInstance: Pointer; hFieldInstance: Pointer; **var** Result: AstaFieldType): Err;

Description
Returns the type of the field for a given Field Handle

AstaDataList

function AstaDLGetIndexOfField(refNum: UInt16; hDLInstance: Pointer; hFieldInstance: Pointer; **var** Result: UInt32): Err;

Description
Returns the "index" of the field in Result of a given Field Handle.

AstaDataList

function AstaDLGetRecordsCount(refNum: UInt16; hDLInstance: Pointer; **var** Result: UInt32): Err;

Description
Returns the number of records contained in an AstaDataList in Result. This count usually corresponds to the number of rows requested to be returned from a remote server.

Example
AstaDLGetRecordsCount(reflib,DataList,RecordCount);

AstaDataList

function AstaDLIsEmpty(refNum: UInt16; hDLInstance: Pointer; **var** Result: boolean): Err;

Description

If Result then the AstaDataList is Empty.

The AstaNestedParamList provides a datatructure that works like a DataSet with a row/column abstraction in order to handle SQL result requests from ASTA SkyWire servers with all the data stored as Strings. This makes it easy to use with User Interface Controls. To handle data natively, and a bit more efficiently use DataLists. NestedParamLists are actually an AstaParamList of AstaParamLists with the first ParamList a list of Fields and the rest of the paramlists containing the actual SQL Result sets.

Nested Param List Example

AstaNPLCreate
ASTANPLDestroy
AstaNPLGetField
AstaNPLGetFieldByName
AstaNPLGetFieldCount
AstaNPLGetRecordsCount
AstaNPLIsBOF
AstaNPLisEmpty
AstaNPLIsEOF
AstaNPLNext
AstaNPLPrevious

AstaNestedParamList

function AstaNPLCreate(refNum: UInt16; hPLInstance: Pointer; **var** Result: Pointer): Err;

Description
Creates an AstaNestedParamList in Result from an incoming ParamList from the server in response to PdaToken PdaNestedSQLSelect (1010)

Example

```
AstaNPLCreate(RefLib, ParamList, NPL);
```

AstaNestedParamList

function AstaNPLDestroy(refNum: UInt16; hPLInstance: Pointer): Err;

Description
Disposes of an AstaNestedfParamList previously created with AstaNPLCreate and releases it's resources.

Example

```
AstaNPLDestroy(RefLib, NPL);
```

AstaNestedParamList

function AstaNPLGetField(refNum: UInt16; hPLInstance: Pointer; **Index**: UInt32; **var** Result: Pointer): Err;

Description
Retrieves a Field as an AstaParamValue from a NestedParamList.

Example

```
for i := 0 to ParamCount - 1 do
    begin
  Value := nil;
  ValueLen := 0;
  AstaNPLGetField(RefLib, NPL, i, Fld);
  if (fld <> nil) then
  begin
    AstaPLParamGetAsString(RefLib, fld, Value, ValueLen);
    Inc(ValueLen);
    Value := MemPtrNew(ValueLen);
    AstaPLParamGetAsString(RefLib, fld, Value, ValueLen);
    Params[i] := Value;
  end;
 end;
```

AstaNestedParamList

function AstaNPLGetFieldByName(refNum: UInt16; hPLInstance: Pointer; **Name**: PChar; **var** Result: Pointer): Err;

Description

Retrieves an AstaNestedParamList Field byName into Result.

Example

AstaNestedParamList

function AstaNPLGetFieldCount(refNum: UInt16; hPLInstance: Pointer; **var** Result: UInt32): Err;

Description
Returns the number of Fields in a AstaNestedParamList

Example
AstaNPLGetFieldCount(RefLib, NPL, ParamCount);

AstaNestedParamList

function AstaNPLGetRecordsCount(refNum: UInt16; hPLInstance: Pointer; **var** Result: **Integer**): Err;

Description

Returns the number of rows in an AstaNestedParamList where each row is an AstaParamList

Example
AstaNPLGetRecordsCount(RefLib, NPL, Count);

AstaNestedParamList

function AstaNPLIsBOF(refNum: UInt16; hPLInstance: Pointer; **var** Result: boolean): Err;

Description

Returns whether the postion of the AstaNestedParamList is Beginning of File similar to the TDataSet.BOF Call

AstaNestedParamList

function AstaNPLIsEmpty(refNum: UInt16; hPLInstance: Pointer; **var** Result: boolean): Err;

Description
Checks to see if an AstaNestedParamList is Empty and returns a boolean in Var Result if it is.

Example

AstaNPLIsEmpty(RefLib, NPL, BoolRes);

AstaNestedParamList

function AstaNPLIsEOF(refNum: UInt16; hPLInstance: Pointer; **var** Result: boolean): Err;

Description

Returns whether the postion of the AstaNestedParamList is End of File similar to the TDataSet.EOF Call

Example

AstaNestedParamList

function AstaNPLNext(refNum: UInt16; hPLInstance: Pointer; **var** Result: boolean): Err;

Description

Moves to the next embedded AstaParamList from an AstaNestedParamlist

AstaNestedParamList

function AstaNPLPrevious(refNum: UInt16; hPLInstance: Pointer; **var** Result: boolean): Err;

Description

Moves to the previous embedded AstaParamList from an AstaNestedParamlist

The ASTA Unified Messaging Initiative allows for cross platform messaging by using AstaParamLists which are easy to use and fully streamable Data containers that can be transported between ASTA clients and servers and between ASTA clients regardless of hardware or operating system.

AstaPLDestroy
AstaPLClear

AstaParamList

function AstaPLClear(refNum: UInt16; hPLInstance: Pointer): Err;

Description

Clears the ParamList of all items.

Example

AstaPLClear(reflib,AstaParamList);

AstaParamList

function AstaPLCopyTo(refNum: UInt16; hPLInstance: Pointer; hDestPLInstance: Pointer): Err;

Description

Copes one AstaParamList to another.

AstaParamList

function AstaPLCreate(refNum: UInt16; InitialSize: Int32; **var** Result: Pointer): Err;

Description

Creates an AstaParamList

Example
This creates an AstaParamlist with 5 initial ParamItems.

```
var
 ParamList:Pointer;

AstaPLCreate(reflib,5,ParamList);
```

AstaParamList

function AstaPLCreateTransportList(refNum: UInt16; hPLInstance: Pointer; **var** Buffer: Pointer; **var** BufLen: **integer**): Err;

Description
Used internally by AstaLIb.prc to stream an AstaParamList.

AstaParamList

function AstaPLDestroy(refNum: UInt16; hPLInstance: Pointer): Err;

Description

Destroys an AstaParamList and releases resources.

Example

AstaParamList

function AstaPLGetCount(refNum: UInt16; hPLInstance: Pointer; **var** Result: UInt32): Err;

Description

Returns the number of items in the AstaParamList

Example
AstaPLGetCount(reflib,ParamList,Count);

AstaParamList

function AstaPLGetParameter(refNum: UInt16; hPLInstance: Pointer; **Index**: UInt32; **var** Result: Pointer): Err;

Description

Retrieves an AstaParamValue at Index from an AstaParamList

Example
var
ParamList : pointer;
ParamItem:Pointer;

AstaPlGetParameter(RefLib, ParamList, 0, ParamItem);

AstaParamList

function AstaPLSetBlob(refNum: UInt16; hPLInstance: Pointer; **Index**: UInt32; value: Pointer; ValueLen: UInt32; **Name**:
PChar): Err;

Description

Sets an AstaParamValue with a buffer of Value of Length ValueLen.

Example

AstaPlSetBlob(RefLib,Params,0,TheBlob,BlobSize,'Blob');

AstaParamList

function AstaPLSetBoolean(refNum: UInt16; hPLInstance: Pointer; **Index**: UInt32; value: boolean; **Name**: PChar): Err;

Description

Sets an AstaParamValue at Index to to a Boolean value with ParamName Name.

AstaParamList

function AstaPLSetByte(refNum: UInt16; hPLInstance: Pointer; **Index**: UInt32; Value: char; **Name**: PChar): Err;

Description

Sets an AstaParamValue at Index to a Byte value with ParamName Name.

Example

AstaParamList

function AstaPLSetDate(refNum: UInt16; hPLInstance: Pointer; **Index**: UInt32; Value: DateTimePtr; **Name**: PChar): Err;

Description

Sets an AstaParamValue at Index to a DateType Value with ParamName Name.

Example

AstaParamList

function AstaPLSetDateTime(refNum: UInt16; hPLInstance: Pointer; **Index**: UInt32; Value: DateTimePtr; **Name**: PChar): Err;

Description

Sets an AstaParamValue at Index to a DateTime value with ParamName Name.

AstaParamList

function AstaPLSetDouble(refNum: UInt16; hPLInstance: Pointer; **Index**: UInt32; value: double; **Name**: PChar): Err;

Description

Sets a Param Item at Index to to a double value with ParamName Name.

Example
AstaPlSetDouble(RefLib,Params,1,5.44,'Double');

AstaParamList

function AstaPLSetFromTransportList(refNum: UInt16; hPLInstance: Pointer; Buf: Pointer; BufLen: UInt32): Err;

Description
Used internally to receive a raw buffer and convert to an AstaParamList.

AstaParamList

function AstaPLSetLongInt(refNum: UInt16; hPLInstance: Pointer; **Index**: UInt32; value: Int32; **Name**: PChar): Err;

Description
Sets an AstaParamValue at Index to LongInt with a ParamName of Name.

Example

```
AstaPlSetLongInt(RefLib,Params,1,45,'Age');
```

AstaParamList

function AstaPLSetNull(refNum: UInt16; hPLInstance: Pointer; **Index**: UInt32; fType: AstaFieldType; **Name**: PChar): Err;

Description

Sets an AstaParamValue to Null.

Example

```
AstaPLSetNull(RefLib,Params,1,pftString,'Department');
```

AstaParamList

function AstaPLSetSingle(refNum: UInt16; hPLInstance: Pointer; **Index**: UInt32; value: single; **Name**: PChar): Err;

Description
Sets a Param Item at Index to to a Single value with ParamName Name.

Example

AstaParamList

function AstaPLSetSmallInt(refNum: UInt16; hPLInstance: Pointer; **Index**: UInt32; value: Int32; **Name**: PChar): Err;

Description

Sets an AstaParamValue at Index to a SmallInt value with ParamName Name.

Example

AstaParamList

function AstaPLSetString(refNum: UInt16; hPLInstance: Pointer; **Index**: UInt32; Value: PChar; **Name**: PChar): Err;

Description

Sets a String AstaParamValue at index with ParamName of Name

Example

```
AstaPlSetString(RefLib, ParamList, 0, 'Select * from Customers', 'PalmSelect');
```

AstaParamList

function AstaPLSetTime(refNum: UInt16; hPLInstance: Pointer; **Index**: UInt32; Value: DateTimePtr; **Name**: PChar): Err;

Description

Sets a Time AstaParamValue at index with ParamName of Name

Example

AstaParamValues are the items contained in the expandable AstaParamLists explained more fully in the ASTA Unified Messaging Initiative

Parameter name - the name of the parameter
Parameter type - the mode of the parameter, in, out, inout, result.
Data type - AstaFieldType which follows the database types (char, bit, integer, etc.)
IsNull flag - a flag indicating if the data is null (a bit representing true or false)
Bytes of data - the actual data as a list of bytes

AstaPLparamGetAsBlob
AstaPLParamGetAsBoolean
AstaPLParamGetAsDate
AstaPLParamGetAsDateTime
AstaPLParamGetAsDouble
AstaPLParam.GetAsLongInt
ASTAPLParamGetAsString
AstaPLParamGetAsStringC
AstaPLParamGetAsTime
AstaPLParamGetName
AstaPLParamGetType
AstaPLParamGetIsNull
AstaPLParamSetName

AstaFieldType=(pftUnknown, pftByte, pftSmallint, pftLongint, pftBoolean, pftSingle, pftDouble, pftDate, pftTime, pftDateTime, pftString, pftBlob);
Description

Value	Meaning
pftUnknown	
pftByte	
pftSmallint	
pftLongint	
pftBoolean	
pftSingle	
pftDouble	
pftDate	
pftTime	
pftDateTime	
pftString	
pftBlob	

AstaParamValue

function AstaPLGetParameterByName(refNum: UInt16; hPLInstance: Pointer; **Name**: PChar; **var** Result: Pointer): Err;

Description

Retrieves an AstaParamValue by ParamName from an AstaParamList

Example

AstaParamValue

function AstaPLGetParameterIndex(refNum: UInt16; hPLInstance: Pointer; hParamInstance: Pointer; **var** Result: UInt32): Err;

Description

Returns the Index of an AstaParamValue from an AstaParamList

Example

AstaParamValue

function AstaPLParamGetAsBlob(refNum: UInt16; hInstance: Pointer; Buffer: Pointer; **var** BufLen: UInt32): Err;

Description

Copies an AstaParamValue Data into Buffer and returns the size of the Blob in BufLen.

AstaParamValue

function AstaPLParamGetAsBoolean(refNum: UInt16; hInstance: Pointer; **var** Result: boolean): Err;

Description

Retrieves the Boolean value of an AstaParamValue in Result:Boolean.

AstaParamValue

function AstaPLParamGetAsDate(refNum: UInt16; hInstance: Pointer; Date: DateTimePtr; **var** Result: boolean): Err;

Description

Retrieves the DateTimePtr value of an AstaParamValue in Date and if successful returns True in Result.

AstaParamValue

function AstaPLParamGetAsDateTime(refNum: UInt16; hInstance: Pointer; Time: DateTimePtr; **var** Result: boolean): Err;

Description

Retrieves the DateTimePtr value of an AstaParamValue in Time and if successful returns True in Result.

AstaParamValue

function AstaPLParamGetAsDouble(refNum: UInt16; hInstance: Pointer; **var** Result: double): Err;

Description

Retrieves a double value of an AstaParamValue in Result.

AstaParamValue

function AstaPLParamGetAsLongInt(refNum: UInt16; hInstance: Pointer; **var** Result: Int32): Err;

Description

Retrieves a LongInt value of an AstaParamValue in Result.

AstaParamValue

function AstaPLParamGetAsString(refNum: UInt16; hInstance: Pointer; NameBuf: PChar; **var** BufLen: UInt32): Err;

Description
Retrieves the AstaParamValue into NameBuf and the length of the Pchar into BufLen.

AstaParamValue

function AstaPLParamGetAsStringC(refNum: UInt16; hInstance: Pointer; NameBuf: PChar): Err;

Description
Retrieves the AstaParamValue into NameBuf

AstaParamValue

function AstaPLParamGetAsTime(refNum: UInt16; hInstance: Pointer; Time: DateTimePtr; **var** Result: boolean): Err;

Description

AstaParamValue

function AstaPLParamGetIsNull(refNum: UInt16; hInstance: Pointer; **var** Result: boolean): Err;

Description
Returns True to Result of the if the ParamValue is null.

AstaParamValue

function AstaPLParamGetName(refNum: UInt16; hInstance: Pointer; NameBuf: PChar; **var** BufLen: UInt32): Err;

Description
Retrieves the Name of the ParamValue along with the buffer length.

AstaParamValue

function AstaPLParamGetNameC(refNum: UInt16; hInstance: Pointer; NameBuf: PChar): Err;

Description
Retrieves the Name of the ParamValue.

Example

AstaParamValue

function AstaPLParamGetType(refNum: UInt16; hInstance: Pointer; **var** Result: AstaFieldType): Err;

Description

Retrieves a AstaFieldtype value of an AstaParamValue in Result.

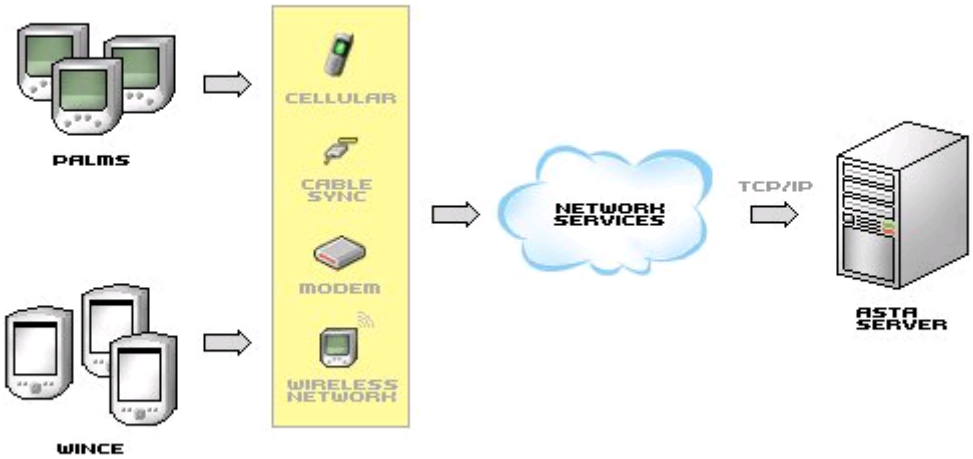
AstaParamValue

function AstaPLParamSetName(refNum: UInt16; hInstance: Pointer; Value: PChar): Err;

Description

Sets an AstaParamValue Name to Value

1.10.2.4.3.5 NSBASIC



The ASTA SkyWire architecture consists of remote thin clients, connecting to a middle tier Application Server via TCP/IP. SkyWire Servers are available using a number of languages include Visual Basic, Delphi and Java. This document describes how to use NSBASIC Clients to access remote SkyWire Servers servers and perform standard tasks using the SkyWire component.

ASTA Skywire consists of a middleware Server and client side components to access that server. The following components are used by SkyWire clients to perform standard tasks.

AstaConnection
AstaParamList
AstaParamValue
AstaDataList
AstaNestedParamList

The AstaClientConnection is the client end of the TCP/IP connection. It connects the ASTA client to the ASTA SkyWire Server. The Address and Port properties on the ASTA client must match the Address and Port properties on the ASTA SkyWire Server. Typically the handheld device would have a form where the IPAddress and Port can be input by the user so that the AstaClientConnection can connect to a SkyWire server and authenticate using a Username and Password.

AstaConnClose
AstaConnCreate
AstaConnDestroy
AstaConnGetFile
AstaConnGetLastError
AstaConnGetUserError
AstaConnOpen
AstaConnReceiveParamList
AstaConnSendParamList
AstaConnSetAuthenticationOff
AstaConnSetAuthenticationOn
AstaConnSetCompressionOff
AstaConnSetCompressionOn
AstaConnSetEncryptionOff
AstaConnSetEncryptionOn
AstaConnSetHttpOn
AstaConnSetHttpOff
AstaConnSetTimeout
AstaRequestUpdate

AstaConnection

function AstaConnClose(refNum: UInt16; hConnInstance: Pointer; **var** Result: boolean): Err;

Description

Closes a connection opened after calling AstaConnOpen.

Example

```
AstaConnClose(RefLib, Conversation, BoolRes);
```

AstaConnection

function AstaConnCreate(refNum: UInt16; **var** Result: Pointer): Err;

Description
Creates an AstaConnection as the Var Result:Pointer where refNum represents the handle to the Asta Shared Library.

Example
```
AstaConnCreate(RefLib, Conversation);
```

AstaConnection

function AstaConnDestroy(refNum: UInt16; hConnInstance: pointer): Err;

Description
Destroys and AstaConnection and releases it's resources that has been created with AstaConnOpen.

Example
```
AstaConnDestroy(RefLib, Conversation);
```

AstaConnection

function AstaConnGetFile(refNum :UInt16; hConnInstance :Pointer; BlockSize:UInt32;
RemoteFile:PChar; LocalName :PChar; Var Result:Boolean):Err

Description
Retrieves a remote file from an ASTA Server in BlockSize "chunks". This allows, for example, for large images to be streamed down efficiently.

Example
AstaConnGetFile(refLib, Connection, 8192, RemoteFileName, LocalFileName, Result);

AstaConnection

function AstaConnGetLastError(refNum: UInt16; hConnInstance: Pointer; **var** Result: Int32): Err;

Description
Returns the last error generated by the AstaClientConnection. AstaSky Wire servers are codedso as to return an Integer value on each client request where a non-zero value is considered to be an error.

Example
AstaConnGetLastError(RefLib,Connection,Result)

AstaConnection

function AstaConnGetUserError(refNum: UInt16; hConnInstance: Pointer; **var** Result: Int32): Err;

escription
Returns the last error generated by the AstaClientConnection. AstaSky Wire servers are coded so as to return an Integer value on each client request where a non-zero value is considered to be an error. Typically errors are also returned with the server side error message or exception in an AstaParamValue of Name "Error"

Example
AstaConnGetUserError(RefLib,Connection,Result);

AstaConnection

function AstaConnOpen(refNum: UInt16; hConnInstance: Pointer; Address: PChar; Port: **Word**; **var** Result: boolean): Err;

Description
Attempts to connect to a remote AstaSkyWire server using the Address and Port and will return a False a connection to the server cannot be achieved. No data is transferred to the server. The Asta SkyWire shared library must already be loaded.

Example

```
AstaConnOpen( RefLib, Conversation, '127.0.01',9050, BoolRes);
```

AstaConnection

function AstaConnReceiveParamList(refNum: UInt16; hConnInstance: Pointer; hPLInstance: Pointer; **var** Result: boolean): Err;

Description
Receives an AstaParamList from a remote ASTA Server where the hPLIntance is an AstaParamList that has been created with a call to AstaPLCreate. If the ParamList was sucessfully retrieved from the server the Var Result:Boolean would be set to True.

Example
Typically, if you had an existing Paramlist that you had used in sending params to a remote server you would want to clear it first.

```
AstaPlClear(RefLib, ParamList);
AstaConnReceiveParamList(RefLib, Conversation, ParamList, BoolRes);
```

AstaConnection

function AstaConnSendParamList(refNum: UInt16; hConnInstance: Pointer; hPLInstance: Pointer; MsgToken: UInt32; **var** Result: boolean): Err;

Description
Sends an AstaParam to a remote Server. The AstaParamList must have been created and typically is filled with values before being sent to the remote server. The MsgToken:UInt32 will determine how the server processes the request. If the ParamList was successfully sent to the server, teh Var Result:Boolean will be set to true.

Example
AstaConnSendParamList(RefLib, Conversation, ParamList, 1001, BoolRes);

AstaConnection

function AstaConnSetAuthorizationOff(refNum: UInt16; hConnInstance: Pointer): Err;

Description
Turns off any Authentication information being sent to the AstaSkyWire server on connect.

Example

```
AstaConnSetAuthorizationOff(RefLib,Conversation);
```

AstaConnection

function AstaConnSetAuthorizationOn(refNum: UInt16; hConnInstance: Pointer; UserName: PChar; Password: PChar; SecurePassword: boolean): Err;

Description
Sets the Username and password to be sent to the remote ASTA SkyWire server to authenticate User Information.

Example

```
AstaConnSetAuthorizationOn(RefLib,Conversation,'ASTA','Password',False);
```

AstaConnection

function AstaConnSetCompressionOff(refNum: UInt16; hConnInstance: Pointer): Err;

Description
Sets Compression off. Requires AstaZLib.prc

Example
```
AstaConnSetCompressionOff(RefLib,Conversation);
```

AstaConnection

function AstaConnSetCompressionOn(refNum: UInt16; hConnInstance: Pointer; level: UInt32): Err;

Description
Sets Compression on where level is the compression factor and the higher the number the greater the compresssion but is also slower. Requires AstaZLib.prc

Example
```
AstaConnSetCompressionOn(RefLib,Conversation,9);
```

AstaConnection

function AstaConnSetEncryptionOff(refNum: UInt16; hConnInstance: Pointer): Err;

Description
Sets AstaAES Encryption off.

Example
AstaConnSetEncryptionOff(RefLib,Connection);

AstaConnection

function AstaConnSetEncryptionOn(refNum: UInt16; hConnInstance: Pointer; key: Pointer; keyMode: UInt32): Err;

Description
Sets EncryptionOn and passes in a Key that is used to encrypt and decrypt the data. The Server must be aware of this key. KeyMode signifies which direction and it should usually set to both (2).

Example
AstaConnSetEncryptionOn(`RefLib,Conversation,'MumsTheWord',2);`

AstaConnection

function AstaConnSetHttpOff(refnum: UInt16; hconnInstance: Pointer): Err

Will be available in Astalib 1.1 and necessary for ASTA Palm SOAP support along with the ASTA Palm

XML Parser

Description
Turns off http tunneling that was set with AstaConnSetHttpOn.

AstaConnection

function AstaConnSetHttpOn(refnum: UInt16; hConnInstance: Pointer; Address: PChar; Port: Word; Page: PChar;
Authenticate: Boolean; UserName, Password: Pchar; UseProxy: Boolean; ProxyAddress: PChar;
ProxyPort: Word; ProxyAuthenticate: Boolean; ProxyUserName: Pchar; ProxyPassWord: Pchar): Err

Will be available in Astalib 1.1 and necessary for ASTA Palm SOAP support along with the ASTA Palm XML Parser

Description
Uses http instead of tcp/ip to allow access through firewalls and through a WebServer using AstaHttp.dll.

Address:Pchar Address of the ASTA Server
Port: Word : Port of the ASTA Server
Authenticate:Boolean': Authenticate at the server
UserName:Pchar UserName
Password: Pchar; Password
UseProxy: Boolean Use a proxy Server
ProxyAddress: PChar; Address of the Proxy Server
ProxyPort: Word Port of the Proxy Server
ProxyAuthenticate: Boolean Autentication required at the Proxy
ProxyUserName: Pchar Username used by Proxy Server
ProxyPassWord: Pchar Password used by Proxy Server

Example

AstaConnection

function AstaConnSetServerVersion(refNum: UInt16; hConnInstance: Pointer; Version: UInt32): Err;

Description
Asta pda libraries can support ASTA for Windows and AstaInterOp for Linux and Windows. This swtich sets the ASTA library.

Valid values are

```
Const
Asta    = 2
AstaIO  = 3;
```

Example

```
AstaConnSetServerVersion(reflib,Connection,Asta);

AstaConnSetServerVersion(reflib,Connection,AstaIO);
```

AstaConnection

function AstaConnSetTimeout(refNum: UInt16; hConnInstance: Pointer; Timeout: UInt32): Err;

Description
Sets the value in Milliseconds that an AstaConnection will "block" in waiting for a response to the server. Since there is no threading on the Palm, sockets must connect to a server and then block or wait.

Example
AstaConnSetTimeout(reflib,Connection,5000);

AstaConnection

function AstaRequestUpdate(refNum: UInt16; hConnInstance: Pointer; Buffer: Pointer; BufLen: UInt32; **var** Result: boolean): Err;

Description

AstaConnection

function AstaSetTerminateConn(refNum: UInt16; hConnInstance: Pointer; value: boolean): Err;

Description

This will the AstaConnection to terminate the connection with the server after each connection. The default is to set this to true and is recommended over any wireless connection. With WinCE clients they may stay connected.

Example

AstaSetTerminateConn(reflib,Connection,True);

The AstaDataList is an efficient DataStructure that handles rows and columns much like a database table. To request a DataList from a remote server you must create an AstaParamList and fill it with the following values:

AstaPLSetString(AstaLibRef, params, 0,'Select * from Customer', "PalmSelect");
AstaPLSetLongInt(AstaLibRef, params, 1, 10000, "RowCount");

Param 0: the Select SQL as a String
Param 1 : the maximum rowcount

The DataList basically contains a List of ParamLists that contain data natively, stored very efficiently, wthout duplicating paramnames, like the AstaNestedParamList does. Each row can be retrieved into a

ParamList with the option of bring the Param or "FieldNames" also by calling AstaDLGetAsParamList

AstaDLCreate
AstaDLDestroy
AstaDLGetAsParamList
AstaDLGetField
AstaDLGetFieldByName
AstaDLGetFieldByNo
AstaDLGetFieldCount
AstaDLGetFieldName
AstaDLGetFieldNameC
AstaDLGetFieldNo
AstaDLGetFieldOptions
AstaDLGetFieldType
AstaDLGetIndexOfField
AstaDLGetRecordsCount
AstaDLIsEmpty

AstaDataList

function AstaDLCreate(refNum: UInt16; Buffer: Pointer; BufLen: UInt32; **var** Result: Pointer): Err;

Description
A DataList is created on a remote ASTA server in response to a select SQL . PdaMultiSQLSelect, or custom result set being streamed back to an ASTA Pda client.

AstaDLCreate Creates a DataList from a paramList AstaParamValue streamed over the wire as a Blob Param.

Example
This examples assumes the ParamList has been received using AstaConnReciveParamList. The ReturnParam:Pointer below is an AstaParamValue.

```
AstaPLGetParameter(refLib, ParamList, 0, ReturnParam);

DataPtr :=nil;
DataLen := 0;
//with Dataptr as NIL the size of the blob is returned in DataLen
AstaPLParamGetAsBlob(RefLib, ReturnParam, DataPtr, DataLen);

DataPtr := MemPtrNew(DataLen+1);//allocate some memory now

AstaPLParamGetAsBlob(Reflib, ReturnParam, DataPtr, DataLen);

DataList:=nil;

AstaDLCreate(Reflib, DataPtr, DataLen, DataList);

MemPtrFree(DataPtr);
```

AstaDataList

function AstaDLDestroy(refNum: UInt16; hDLInstance: Pointer): Err;

Description
Destroys an AstaDataList and frees up it's resources;

Example
AstaDLDestroy(refLib,DataList);

AstaDataList

function AstaDLGetAsParamList(refNum: UInt16; hDLInstance: Pointer; recordNumber: UInt32; putNames: boolean; **var** Result: Pointer): Err;

Description
Retrieves an AstaParamList at the row designated as RecordNumber into the Var Result:Pointer. The number of rows or records can be obtained by calling AstaDLGetRecordsCount.

Example
AstaDLGetAsParamList(refLib,DataList,1,True,TempParamList);

AstaDataList

function AstaDLGetField(refNum: UInt16; hDLInstance: Pointer; **Index**: UInt32; **var** Result: Pointer): Err;

Description
Returns a handle to a Field by a given Index.

AstaDataList

function AstaDLGetFieldByName(refNum: UInt16; hDLInstance: Pointer; **Name**: PChar; **var** Result: Pointer): Err;

Description
Returns the "handle" to the "current" field by given name

AstaDataList

function AstaDLGetFieldByNo(refNum: UInt16; hDLInstance: Pointer; **Index**: UInt32; **var** Result: Pointer): Err;

Description

Returns the "handle" to the field by given index

AstaDataList

function AstaDLGetFieldCount(refNum: UInt16; hDLInstance: Pointer; **var** Result: UInt32): Err;

Description
Returns the number of fields in a DataLis in the Var Result: UInt32.

Example
AstaDLGetFieldCount(refLib,DataList,Fieldcount);

AstaDataList

function AstaDLGetFieldName(refNum: UInt16; hDLInstance: Pointer; hFieldInstance: Pointer; NameBuf: PChar; **var** BufLen: UInt32): Err;

Description
Returns the fieldname of a given field when a Field Handle is passed in as well as the length of the field in BufLen

AstaDataList

function AstaDLGetFieldNameC(refNum: UInt16; hDLInstance: Pointer; hFieldInstance: Pointer; NameBuf: PChar): Err;

Description
Returns the name of the field

AstaDataList

function AstaDLGetFieldNo(refNum: UInt16; hDLInstance: Pointer; hFieldInstance: Pointer; **var** Result: UInt32): Err;

Description
Returns the Nnumber of the field.

AstaDataList

function AstaDLGetFieldOptions(refNum: UInt16; hDLInstance: Pointer; hFieldInstance: Pointer; **var** Result: UInt32): Err;

```
#define pfaHiddenCol    1
#define pfaReadonly     2
#define pfaRequired     4
```

```
#define pfaLink          8
#define pfaUnNamed       16
#define pfaFixed                 3
```

Description
Returns the options of the field.

AstaDataList

function AstaDLGetFieldType(refNum: UInt16; hDLInstance: Pointer; hFieldInstance: Pointer; **var** Result: AstaFieldType): Err;

Description
Returns the type of the field for a given Field Handle

AstaDataList

function AstaDLGetIndexOfField(refNum: UInt16; hDLInstance: Pointer; hFieldInstance: Pointer; **var** Result: UInt32): Err;

Description
Returns the "index" of the field in Result of a given Field Handle.

AstaDataList

function AstaDLGetRecordsCount(refNum: UInt16; hDLInstance: Pointer; **var** Result: UInt32): Err;

Description
Returns the number of records contained in an AstaDataList in Result. This count usually corresponds to the number of rows requested to be returned from a remote server.

Example

```
AstaDLGetRecordsCount(reflib,DataList,RecordCount);
```

AstaDataList

function AstaDLIsEmpty(refNum: UInt16; hDLInstance: Pointer; **var** Result: boolean): Err;

Description

If Result then the AstaDataList is Empty.

The AstaNestedParamList provides a datatructure that works like a DataSet with a row/column abstraction in order to handle SQL result requests from ASTA SkyWire servers with all the data stored as Strings. This makes it easy to use with User Interface Controls. To handle data natively, and a bit more efficiently use DataLists. NestedParamLists are actually an AstaParamList of AstaParamLists with the first ParamList a list of Fields and the rest of the paramlists containing the actual SQL Result sets.

Nested Param List Example

AstaNPLCreate
ASTANPLDestroy
AstaNPLGetField
AstaNPLGetFieldByName
AstaNPLGetFieldCount
AstaNPLGetRecordsCount
AstaNPLIsBOF
AstaNPLisEmpty
AstaNPLIsEOF
AstaNPLNext
AstaNPLPrevious

AstaNestedParamList

function AstaNPLCreate(refNum: UInt16; hPLInstance: Pointer; **var** Result: Pointer): Err;

Description
Creates an AstaNestedParamList in Result from an incoming ParamList from the server in response to PdaToken PdaNestedSQLSelect (1010)

Example

```
AstaNPLCreate(RefLib, ParamList, NPL);
```

AstaNestedParamList

function AstaNPLDestroy(refNum: UInt16; hPLInstance: Pointer): Err;

Description
Disposes of an AstaNestedfParamList previously created with AstaNPLCreate and releases it's resources.

Example

```
AstaNPLDestroy(RefLib, NPL);
```

AstaNestedParamList

function AstaNPLGetField(refNum: UInt16; hPLInstance: Pointer; **Index**: UInt32; **var** Result: Pointer): Err;

Description
Retrieves a Field as an AstaParamValue from a NestedParamList.

Example

```
for i := 0 to ParamCount - 1 do
    begin
  Value := nil;
  ValueLen := 0;
  AstaNPLGetField(RefLib, NPL, i, Fld);
  if (fld <> nil) then
  begin
    AstaPLParamGetAsString(RefLib, fld, Value, ValueLen);
    Inc(ValueLen);
    Value := MemPtrNew(ValueLen);
    AstaPLParamGetAsString(RefLib, fld, Value, ValueLen);
    Params[i] := Value;
  end;
 end;
```

AstaNestedParamList

function AstaNPLGetFieldByName(refNum: UInt16; hPLInstance: Pointer; **Name**: PChar; **var** Result: Pointer): Err;

Description

Retrieves an AstaNestedParamList Field byName into Result.

Example

AstaNestedParamList

function AstaNPLGetFieldCount(refNum: UInt16; hPLInstance: Pointer; **var** Result: UInt32): Err;

Description
Returns the number of Fields in a AstaNestedParamList

Example
AstaNPLGetFieldCount(RefLib, NPL, ParamCount);

AstaNestedParamList

function AstaNPLGetRecordsCount(refNum: UInt16; hPLInstance: Pointer; **var** Result: **Integer**): Err;

Description

Returns the number of rows in an AstaNestedParamList where each row is an AstaParamList

Example
AstaNPLGetRecordsCount(RefLib, NPL, Count);

AstaNestedParamList

function AstaNPLIsBOF(refNum: UInt16; hPLInstance: Pointer; **var** Result: boolean): Err;

Description

Returns whether the postion of the AstaNestedParamList is Beginning of File similar to the TDataSet.BOF Call

AstaNestedParamList

function AstaNPLIsEmpty(refNum: UInt16; hPLInstance: Pointer; **var** Result: boolean): Err;

Description
Checks to see if an AstaNestedParamList is Empty and returns a boolean in Var Result if it is.

Example

AstaNPLIsEmpty(RefLib, NPL, BoolRes);

AstaNestedParamList

function AstaNPLIsEOF(refNum: UInt16; hPLInstance: Pointer; **var** Result: boolean): Err;

Description

Returns whether the postion of the AstaNestedParamList is End of File similar to the TDataSet.EOF Call

Example

AstaNestedParamList

function AstaNPLNext(refNum: UInt16; hPLInstance: Pointer; **var** Result: boolean): Err;

Description

Moves to the next embedded AstaParamList from an AstaNestedParamlist

AstaNestedParamList

function AstaNPLPrevious(refNum: UInt16; hPLInstance: Pointer; **var** Result: boolean): Err;

Description

Moves to the previous embedded AstaParamList from an AstaNestedParamlist

The ASTA Unified Messaging Initiative allows for cross platform messaging by using AstaParamLists which are easy to use and fully streamable Data containers that can be transported between ASTA clients and servers and between ASTA clients regardless of hardware or operating system.

AstaPLDestroy
AstaPLClear
AstaPLCopyTo
AstaPLCreate
AstaPLCreateTransportList
AstaPLGetCount
AstaPLGetParameter
AstaPLGetParameterByName
AstaPLGetParameterIndex
AstaPLSetBlob
AstaPLSetBoolean
AstaPLSetByte
AstaPLSetDate
AstaPLSetDateTime
AstaPLSetDouble

AstaPLSetFromTransportList
AstaPLSetLongInt
AstaPLSetNull
AstaPLSetSingle
AstaPLSetSmallInt
AstaPLSetString
AstaPLSetTime

AstaParamList

function AstaPLClear(refNum: UInt16; hPLInstance: Pointer): Err;

Description

Clears the ParamList of all items.

Example

AstaPLClear(reflib,AstaParamList);

AstaParamList

function AstaPLCopyTo(refNum: UInt16; hPLInstance: Pointer; hDestPLInstance: Pointer): Err;

Description

Copes one AstaParamList to another.

AstaParamList

function AstaPLCreate(refNum: UInt16; InitialSize: Int32; **var** Result: Pointer): Err;

Description

Creates an AstaParamList

Example
This creates an AstaParamlist with 5 initial ParamItems.

```
var
 ParamList:Pointer;

AstaPLCreate(reflib,5,ParamList);
```

AstaParamList

function AstaPLCreateTransportList(refNum: UInt16; hPLInstance: Pointer; **var** Buffer: Pointer; **var** BufLen: **integer**): Err;

Description
Used internally by AstaLIb.prc to stream an AstaParamList.

AstaParamList

function AstaPLDestroy(refNum: UInt16; hPLInstance: Pointer): Err;

Description

Destroys an AstaParamList and releases resources.

Example

AstaParamList

function AstaPLGetCount(refNum: UInt16; hPLInstance: Pointer; **var** Result: UInt32): Err;

Description

Returns the number of items in the AstaParamList

Example
AstaPLGetCount(reflib,ParamList,Count);

AstaParamList

function AstaPLGetParameter(refNum: UInt16; hPLInstance: Pointer; **Index**: UInt32; **var** Result: Pointer): Err;

Description

Retrieves an AstaParamValue at Index from an AstaParamList

Example
var
ParamList : pointer;
ParamItem:Pointer;

AstaPlGetParameter(RefLib, ParamList, 0, ParamItem);

AstaParamList

function AstaPLSetBlob(refNum: UInt16; hPLInstance: Pointer; **Index**: UInt32; value: Pointer; ValueLen: UInt32; **Name**:

PChar): Err;

Description

Sets an AstaParamValue with a buffer of Value of Length ValueLen.

Example

AstaPlSetBlob(RefLib,Params,0,TheBlob,BlobSize,'Blob');

AstaParamList

function AstaPLSetBoolean(refNum: UInt16; hPLInstance: Pointer; **Index**: UInt32; value: boolean; **Name**: PChar): Err;

Description

Sets an AstaParamValue at Index to to a Boolean value with ParamName Name.

AstaParamList

function AstaPLSetByte(refNum: UInt16; hPLInstance: Pointer; **Index**: UInt32; Value: char; **Name**: PChar): Err;

Description

Sets an AstaParamValue at Index to a Byte value with ParamName Name.

Example

AstaParamList

function AstaPLSetDate(refNum: UInt16; hPLInstance: Pointer; **Index**: UInt32; Value: DateTimePtr; **Name**: PChar): Err;

Description

Sets an AstaParamValue at Index to a DateType Value with ParamName Name.

Example

AstaParamList

function AstaPLSetDateTime(refNum: UInt16; hPLInstance: Pointer; **Index**: UInt32; Value: DateTimePtr; **Name**: PChar): Err;

Description

Sets an AstaParamValue at Index to a DateTime value with ParamName Name.

AstaParamList

function AstaPLSetDouble(refNum: UInt16; hPLInstance: Pointer; **Index**: UInt32; value: double; **Name**: PChar): Err;

Description

Sets a Param Item at Index to to a double value with ParamName Name.

Example
AstaPlSetDouble(RefLib,Params,1,5.44,'Double');

AstaParamList

function AstaPLSetFromTransportList(refNum: UInt16; hPLInstance: Pointer; Buf: Pointer; BufLen: UInt32): Err;

Description
Used internally to receive a raw buffer and convert to an AstaParamList.

AstaParamList

function AstaPLSetLongInt(refNum: UInt16; hPLInstance: Pointer; **Index**: UInt32; value: Int32; **Name**: PChar): Err;

Description
Sets an AstaParamValue at Index to LongInt with a ParamName of Name.

Example

AstaPlSetLongInt(RefLib,Params,1,45,'Age');

AstaParamList

function AstaPLSetNull(refNum: UInt16; hPLInstance: Pointer; **Index**: UInt32; fType: AstaFieldType; **Name**: PChar): Err;

Description

Sets an AstaParamValue to Null.

Example
AstaPLSetNull(RefLib,Params,1,pftString,'Department');

AstaParamList

function AstaPLSetSingle(refNum: UInt16; hPLInstance: Pointer; **Index**: UInt32; value: single; **Name**: PChar): Err;

Description
Sets a Param Item at Index to to a Single value with ParamName Name.

Example

AstaParamList

function AstaPLSetSmallInt(refNum: UInt16; hPLInstance: Pointer; **Index**: UInt32; value: Int32; **Name**: PChar): Err;

Description

Sets an AstaParamValue at Index to a SmallInt value with ParamName Name.

Example

AstaParamList

function AstaPLSetString(refNum: UInt16; hPLInstance: Pointer; **Index**: UInt32; Value: PChar; **Name**: PChar): Err;

Description

Sets a String AstaParamValue at index with ParamName of Name

Example

AstaPlSetString(RefLib, ParamList, 0, 'Select * from Customers', 'PalmSelect');

AstaParamList

function AstaPLSetTime(refNum: UInt16; hPLInstance: Pointer; **Index**: UInt32; Value: DateTimePtr; **Name**: PChar): Err;

Description

Sets a Time AstaParamValue at index with ParamName of Name

Example

AstaParamValues are the items contained in the expandable AstaParamLists explained more fully in the ASTA Unified Messaging Initiative

Parameter name - the name of the parameter
Parameter type - the mode of the parameter, in, out, inout, result.
Data type - AstaFieldType which follows the database types (char, bit, integer, etc.)
IsNull flag - a flag indicating if the data is null (a bit representing true or false)
Bytes of data - the actual data as a list of bytes

AstaPLparamGetAsBlob
AstaPLParamGetAsBoolean
AstaPLParamGetAsDate
AstaPLParamGetAsDateTime
AstaPLParamGetAsDouble
AstaPLParam.GetAsLongInt
ASTAPLParamGetAsString
AstaPLParamGetAsStringC
AstaPLParamGetAsTime
AstaPLParamGetName
AstaPLParamGetType
AstaPLParamGetIsNull
AstaPLParamSetName

AstaFieldType=(pftUnknown, pftByte, pftSmallint, pftLongint, pftBoolean, pftSingle, pftDouble, pftDate, pftTime, pftDateTime, pftString, pftBlob);
Description

Value	**Meaning**
pftUnknown	
pftByte	
pftSmallint	
pftLongint	
pftBoolean	
pftSingle	
pftDouble	
pftDate	
pftTime	
pftDateTime	
pftString	
pftBlob	

AstaParamValue

function AstaPLGetParameterByName(refNum: UInt16; hPLInstance: Pointer; **Name**: PChar; **var** Result: Pointer): Err;

Description

Retrieves an AstaParamValue by ParamName from an AstaParamList

Example

AstaParamValue

function AstaPLGetParameterIndex(refNum: UInt16; hPLInstance: Pointer; hParamInstance: Pointer; **var** Result: UInt32): Err;

Description

Returns the Index of an AstaParamValue from an AstaParamList

Example

AstaParamValue

function AstaPLParamGetAsBlob(refNum: UInt16; hInstance: Pointer; Buffer: Pointer; **var** BufLen: UInt32): Err;

Description

Copies an AstaParamValue Data into Buffer and returns the size of the Blob in BufLen.

AstaParamValue

function AstaPLParamGetAsBoolean(refNum: UInt16; hInstance: Pointer; **var** Result: boolean): Err;

Description

Retrieves the Boolean value of an AstaParamValue in Result:Boolean.

AstaParamValue

function AstaPLParamGetAsDate(refNum: UInt16; hInstance: Pointer; Date: DateTimePtr; **var** Result: boolean): Err;

Description

Retrieves the DateTimePtr value of an AstaParamValue in Date and if successful returns True in Result.

AstaParamValue

function AstaPLParamGetAsDateTime(refNum: UInt16; hInstance: Pointer; Time: DateTimePtr; **var**

Result: boolean): Err;

Description

Retrieves the DateTimePtr value of an AstaParamValue in Time and if successful returns True in Result.

AstaParamValue

function AstaPLParamGetAsDouble(refNum: UInt16; hInstance: Pointer; **var** Result: double): Err;

Description

Retrieves a double value of an AstaParamValue in Result.

AstaParamValue

function AstaPLParamGetAsLongInt(refNum: UInt16; hInstance: Pointer; **var** Result: Int32): Err;

Description

Retrieves a LongInt value of an AstaParamValue in Result.

AstaParamValue

function AstaPLParamGetAsString(refNum: UInt16; hInstance: Pointer; NameBuf: PChar; **var** BufLen: UInt32): Err;

Description
Retrieves the AstaParamValue into NameBuf and the length of the Pchar into BufLen.

AstaParamValue

function AstaPLParamGetAsStringC(refNum: UInt16; hInstance: Pointer; NameBuf: PChar): Err;

Description
Retrieves the AstaParamValue into NameBuf

AstaParamValue

function AstaPLParamGetAsTime(refNum: UInt16; hInstance: Pointer; Time: DateTimePtr; **var** Result: boolean): Err;

Description

AstaParamValue

function AstaPLParamGetIsNull(refNum: UInt16; hInstance: Pointer; **var** Result: boolean): Err;

Description
Returns True to Result of the if the ParamValue is null.

AstaParamValue

function AstaPLParamGetName(refNum: UInt16; hInstance: Pointer; NameBuf: PChar; **var** BufLen: UInt32): Err;

Description
Retrieves the Name of the ParamValue along with the buffer length.

AstaParamValue

function AstaPLParamGetNameC(refNum: UInt16; hInstance: Pointer; NameBuf: PChar): Err;

Description
Retrieves the Name of the ParamValue.

Example

AstaParamValue

function AstaPLParamGetType(refNum: UInt16; hInstance: Pointer; **var** Result: AstaFieldType): Err;

Description

Retrieves a AstaFieldtype value of an AstaParamValue in Result.

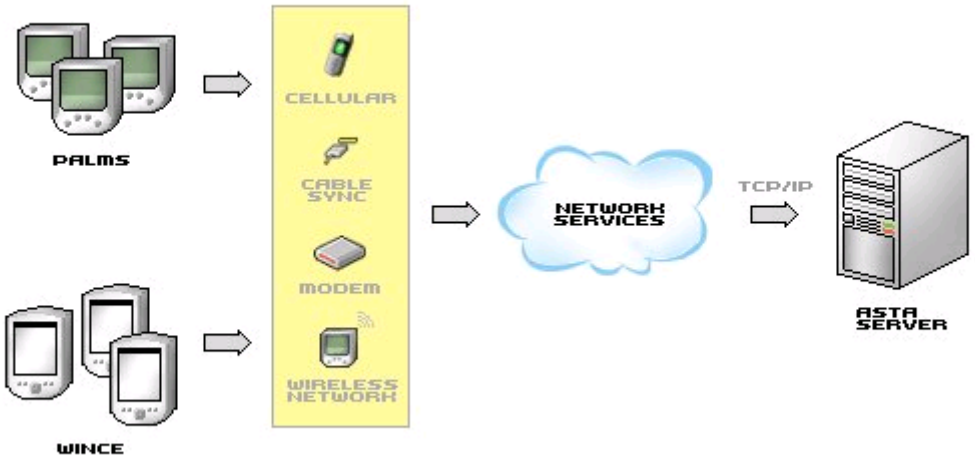
AstaParamValue

function AstaPLParamSetName(refNum: UInt16; hInstance: Pointer; Value: PChar): Err;

Description

Sets an AstaParamValue Name to Value

1.10.2.4.3.6 SkyWire Visual Basic Client



The ASTA SkyWire architecture consists of remote thin clients, connecting to a middle tier Application Server via TCP/IP. SkyWire Servers are available using a number of languages include Visual Basic, Delphi and Java. This document describes how to use SkyWire Clients to access remote SkyWire Servers servers and perform standard tasks using the SkyWire component via AstaComClient.dll available on Win32 or WinCE.

AstaCOMClient.dll is the DLL must be registered before it can be used. Under Win32 most install programs provide a facility to register the DLL or you can manually call

RegSvr32.exe AstaCOMClient.dll

On the WinCE Emulator Regsvrce.exe can be called
RegSvrCE.exe AstaCOMClient.dll

```
Proc = GetProcAddress(LoadLibrary('AstaCOMClient.dll'), 'DLLRegisterServer');
Proc();
```

ASTA Skywire consists of a middleware Server and client side components to access that server. The following components are used by SkyWire clients to perform standard tasks. The client is implemented as AstaCOMClient.dll which is available on Win32 for Visual Basic or from WinCE using Embedded Visual Basic (eVB) or any WinCE development tool capable of making a call to a COM DLL.

CAstaConnection
CAstaParamlist
CAstaParamItem
CAstaDataList
CAstaNestedParamList

The CAstaConnection is the client end of the TCP/IP connection. It connects the ASTA client to the ASTA SkyWire Server. The Host and Port properties on the ASTA client must match the Address and Port properties on the ASTA SkyWire Server. Typically the handheld device would have a form where the IPAddress and Port can be input by the user so that the AstaClientConnection can connect to a SkyWire server and authenticate using a Username and Password.

Active
Compression
Connect
Disconnect
Encryption
Host
Password
Port
SendGetCodedParamList
SendGetDataList
UserName

CAstaConnection

```
Property Active As Boolean   Read-only    Member of AstaCOMClient.CAstaConnection    Property Active
```

Description
Returns whether the CAstaConnection is connected to a remote server.

CAstaConnection

```
Property Compression As Boolean    Member of AstaCOMClient.CAstaConnection    Property
Compression
```

Description
Sets compression on or off. The server must use the same setting.

CAstaConnection

```
Function Connect() As Boolean
    Member of AstaCOMClient.CAstaConnection
    method Connect
```

Description
Attempts to connect to a remote AstaSkyWire server using the Address and Port and will return a False a connection to the server cannot be achieved. No data is transferred to the server. You can check to see if the CAstaConnection using the Active property

CAstaConnection

```
Sub Disconnect()    Member of AstaCOMClient.CAstaConnection    method Disconnect
```

Description

Closes a connection opened after calling Connect

CAstaConnection

```
Property Encryption As Long    Member of AstaCOMClient.CAstaConnection    Property Encryption
```

Description

CAstaConnection

```
Function GetFile(BlockSize As Long, RequestString As String, LocalFile As String) As Long
    Member of AstaCOMClient.CAstaConnection
    method GetFile
```

Description
Allows a file to be retrieved from a remote server in configurable "chunks" as represented by BlockSize. A value of 8192 is an appropriate default value. The RequestString represents the filename on the server and it will be saved as the LocalFile String.

CAstaConnection

```
Property Host As String    Member of AstaCOMClient.CAstaConnection    Property Host
```

Description
Along with the Port property the Host makes up a socket that can connect to a remote SkyWire Server running at the Host address on the specific Port.

CAstaConnection

```
Property LastError As Long
    Read-only
    Member of AstaCOMClient.CAstaConnection
    Property LastError
```

Description
Returns the last error generated by the CAstaConnection. AstaSky Wire servers are coded so as to return an Integer value on each client request where a non-zero value is considered to be an error.

CAstaConnection

```
Property Password As String    Member of AstaCOMClient.CAstaConnection    Property Password
```

Description
Sets the Password that is used along with the UserName property to perform Authentication at the remote SkyWire server.
.

CAstaConnection

```
Property Port As Long    Member of AstaCOMClient.CAstaConnection    Property Port
```

Description
The Port and Host property make up a Socket which allows a client to connect to a remote ASTA SkyWire server running at the address presented by Host on the same Port.

CAstaConnection

```
Function SendGetDatalist(MsgID As Integer, Paramlist As CAstaParamlist) As CAstaDatalist
Member of AstaCOMClient.CAstaConnection    method SendGetDatalist
```

Description
ASTA clients can request row/column information that are simliar to RecordSets. SendGetDataList returns a CAstaDataList.

CAstaConnection

```
Function SendGetNestedParamlist(MsgID As Integer, Paramlist As CAstaParamlist) As
CAstaNestedParamlist    Member of AstaCOMClient.CAstaConnection    method
SendGetNestedParamlist
```

Description
Receives a CAstaNestedParamList from a remote ASTA Server.

CAstaConnection

```
unction SendGetParamlist(MsgID As Integer, Paramlist As CAstaParamlist) As CAstaParamlist
Member of AstaCOMClient.CAstaConnection    method SendGetParamlist
```

Description
Receives a CAstaParamlist from a remote ASTA Server.

Example
Typically, if you had an existing Paramlist that you had used in sending params to a remote server you would want to clear it first.

CAstaConnection

```
Property ServerVersion As Integer
    Member of AstaCOMClient.CAstaConnection
    Property ServerVersion
```

Description
Asta pda libraries can support ASTA for Windows and AstaInterOp for Linux and Windows. This swtich sets the ASTA library.

Valid values are

Asta = 2

AstaIO = 3

CAstaConnection

```
Sub SetAuthenticateOff()
    Member of AstaCOMClient.CAstaConnection
    method SetAuthenticateOff
```

Description
Turns off any Authentication information being sent to the AstaSkyWire server on connect.

CAstaConnection

```
Sub SetAuthenticateOn(UserName As String, Password As String, SecurePassword As Long)
    Member of AstaCOMClient.CAstaConnection
    method SetAuthenticateOn
```

Description
Sets the Username and password to be sent to the remote ASTA SkyWire server to authenticate User Information.

CAstaConnection

```
Property TerminateConn As Long
    Member of AstaCOMClient.CAstaConnection
    Property TerminateConn
```

Description
This will the CAstaConnection to terminate the connection with the server after each connection. The default is to set this to true and is recommended over any wireless connection.

CAstaConnection

```
Property TimeOut As Long
    Member of AstaCOMClient.CAstaConnection
    Property TimeOut
```

Description
Sets the value in Milliseconds that an AstaConnection will "block" in waiting for a response to the server.

CAstaConnection

```
Property UserError As Long
    Read-only
    Member of AstaCOMClient.CAstaConnection
    Property UserError
```

Description
Returns the last error generated by the AstaClientConnection. AstaSky Wire servers are coded so as to return an Integer value on each client request where a non-zero value is considered to be an error. Typically errors are also returned with the server side error message or exception in an AstaParamValue of Name "Error"

CAstaConnection

```
Property UserName As String    Member of AstaCOMClient.CAstaConnection    Property UserName
```

Description
Sets the UserName that is used along with the Password property to perform Authentication at the remote SkyWire server.

The CAstaDataList is an efficient DataStructure that handles rows and columns much like a database recordset.

BOF
Empty
Field
FieldByName
FieldCount
GetParamList
MoveFirst
MoveLast
MoveNext
MovePrevious
MoveTo
Position
RecordCount

CAstaDataList

```
Property BOF As Boolean
    Read-only
    Member of AstaCOMClient.CAstaDatalist
    Property BOF
```

Description
Returns True of the CAstaDataList is positioned at Beginning of File.

CAstaDataList

```
Property Empty As Boolean    Read-only    Member of AstaCOMClient.CAstaDatalist    Property
Empty
```

Description

If Result then the CAstaDataList is Empty.

CAstaDataList

```
Property EOF As Boolean
    Read-only
    Member of AstaCOMClient.CAstaDataParamlist
    Property EOF
```

Description
Returns True of the CAstaDataList is positioned at the End of File.

CAstaDataList

```
Function Field(Index As Long) As CAstaParamitem    Member of AstaCOMClient.CAstaDatalist
method Field
```

Description
Returns a handle to a Field by a given Index.

CAstaDataList

```
Function FieldByName(Name As String) As CAstaParamitem    Member of
AstaCOMClient.CAstaDatalist    method FieldByName
```

Description
Returns the CAstaParamItem current field by given name

CAstaDataList

```
Property FieldCount As Long    Read-only    Member of AstaCOMClient.CAstaDatalist    Property
FieldCount
```

Description
Returns the number of fields in the CAstaDataList.

CAstaDataList

```
Function GetParamlist() As CAstaParamlist    Member of AstaCOMClient.CAstaDatalist    method
GetParamList
```

Description
Retrieves an CAstaParamlist from the current row.

CAstaDataList

```
Sub MoveFirst()
    Member of AstaCOMClient.CAstaDatalist
    method MoveFirst
```

Description
Moves to the first row in the CAstaDataList.

CAstaDataList

```
Sub MoveLast()
    Member of AstaCOMClient.CAstaDatalist
    method MoveLast
```

Description
Moves to the last row in the CAstaDataList.

CAstaDataList

```
Sub MoveNext()    Member of AstaCOMClient.CAstaDatalist    method MoveNext
```

Description
Moves to the next row in the CAstaDataList.

CAstaDataList

```
Sub MovePrevious()
    Member of AstaCOMClient.CAstaDatalist
    method MovePrevious
```

Description
Moves to the previous row in the CAstaDataList.

CAstaDataList

```
Sub MoveTo(Index As Long)
    Member of AstaCOMClient.CAstaDatalist
    method MoveTo
```

Description
Moves to a specific row position in the CAstaDataList.

CAstaDataList

```
Property Position As Long
    Read-only
    Member of AstaCOMClient.CAstaDatalist
    Property Position
```

Description
Returns the row position of the CAstaDataList.

CAstaDataList

```
Property RecordCount As Long    Read-only    Member of AstaCOMClient.CAstaDatalist    Property
RecordCount
```

Description
Returns the number of records contained in an CAstaDataList in Result. This count usually corresponds to the number of rows requested to be returned from a remote server.

The CAstaNestedParamList provides a datatructure that works like a DataSet with a row/column abstraction in order to handle SQL result requests from ASTA SkyWire servers with all the data stored as Strings. This makes it easy to use with User Interface Controls. To handle data natively, and a bit more efficiently use DataLists. NestedParamLists are actually an CAstaParamlist of CAstaParamlists with the first ParamList a list of Fields and the rest of the paramlists containing the actual SQL Result sets.

BOF
Empty
Field
FieldByName
FieldCount
GetParamList
MoveFirst
MoveLast
MoveNext
MovePrevious
MoveTo
Position
RecordCount

CAstaNestedParamList

```
Property BOF As Long    Read-only    Member of AstaCOMClient.CAstaNestedParamlist    Property
BOF
```

Description

Returns whether the postion of the CAstaNestedParamList is Beginning of File similar to the TDataSet.BOF Call

CAstaNestedParamList

```
Property Empty As Long    Read-only    Member of AstaCOMClient.CAstaNestedParamlist
Property Empty
```

Description
Checks to see if an CAstaNestedParamList is Empty and returns a boolean in Var Result if it is.

CAstaNestedParamList

```
Property EOF As Long    Read-only    Member of AstaCOMClient.CAstaNestedParamlist    Property
EOF
```

Description

Returns whether the postion of the CAstaNestedParamList is End of File similar to the TDataSet.EOF Call

CAstaNestedParamList

```
Function Field(Index As Long) As CAstaParamitem    Member of
AstaCOMClient.CAstaNestedParamlist    method Field
```

Description
Returns the CAstaParamItem current field by given name

CAstaNestedParamList

```
Function FieldByName(Name As String) As CAstaParamitem    Member of
AstaCOMClient.CAstaNestedParamlist    method FieldByName
```

Description
Returns the CAstaParamItem current field by given name

CAstaNestedParamList

```
Property FieldCount As Long    Read-only    Member of AstaCOMClient.CAstaNestedParamlist
Property FieldCount
```

Description
Returns the number of Fields in a CAstaNestedParamList

Example
AstaNPLGetFieldCount(RefLib, NPL, ParamCount);

CAstaNestedParamList

```
Function GetParamlist() As CAstaParamlist    Member of AstaCOMClient.CAstaNestedParamlist
method GetParamlist
```

Description
Retrieves an CAstaParamlist from the current row

CAstaNestedParamList

```
Sub MoveFirst()
Member of AstaCOMClient.CAstaDatalist
method MoveFirst
```

Description
Moves to the first row in the CAstaDataList.

CAstaNestedParamList

```
Sub MoveLast()    Member of AstaCOMClient.CAstaNestedParamlist    method MoveLast
```

Description
Moves to the last row in the CAstaDataList.

CAstaNestedParamList

```
Sub MoveNext()    Member of AstaCOMClient.CAstaNestedParamlist    method MoveNext
```

Description

Moves to the next embedded CAstaParamlist from an CAstaNestedParamList

CAstaNestedParamList

```
Sub MovePrevious()    Member of AstaCOMClient.CAstaNestedParamlist    method MovePrevious
```

Description

Moves to the previous embedded CAstaParamlist from an CAstaNestedParamList

CAstaNestedParamList

```
Sub MoveTo(Index As Long)    Member of AstaCOMClient.CAstaNestedParamlist    method MoveTo
```

Description
Moves to a specific row position in the CAstaDataList.

CAstaNestedParamList

```
Property Position As Long    Read-only    Member of AstaCOMClient.CAstaNestedParamlist
Property Position
```

Description
Returns the row position of the CAstaDataList.

CAstaNestedParamList

```
Property RecordCount As Long    Read-only    Member of AstaCOMClient.CAstaNestedParamlist
Property RecordCount
```

Description

Returns the number of rows in an CAstaNestedParamList.

The ASTA Unified Messaging Initiative allows for cross platform messaging by using CAstaParamlists which are easy to use and fully streamable Data containers that can be transported between ASTA clients and servers and between ASTA clients regardless of hardware or operating system.

Add
AddParam
Clear

CAstaParamlist

```
Sub AddParam(Name As String, newVal)    Member of AstaCOMClient.CAstaParamlist    method
AddParam
```

Description
Adds a CAstaParamItem with "Name" using newVal

CAstaParamlist

```
Function Add() As CAstaParamitem    Member of AstaCOMClient.CAstaParamlist    method Add
```

Description
Adds a CAstaParamItem to the list.

CAstaParamlist

```
Function ParamByName(Name As String) As CAstaParamitem    Member of
AstaCOMClient.CAstaParamlist    method ParamByName
```

Description

Retrieves an CAstaParamItem by ParamName from an CAstaParamlist

CAstaParamlist

```
Sub Clear()    Member of AstaCOMClient.CAstaParamlist    method Clear
```

Description

Clears the ParamList of all items.

CAstaParamlist

```
Sub CopyTo(Paramlist As CAstaParamlist)    Member of AstaCOMClient.CAstaParamlist    method
CopyTo
```

Description

Copes one CAstaParamlist to another.

CAstaParamlist

```
Function Count() As Long    Member of AstaCOMClient.CAstaParamlist    method Count
```

Description

Returns the number of items in the CAstaParamlist

CAstaParamlist

```
Function Params(Index As Long) As CAstaParamitem    Member of AstaCOMClient.CAstaParamlist
method Params
```

Description

Retrieves an CAstaParamItem at Index from an CAstaParamlist

CAstaParamItems are the items contained in the expandable CAstaParamlists explained more fully in the ASTA Unified Messaging Initiative

Parameter name	The name of the parameter
Data type	An Integer constant that represents the datatype
IsNull flag	A flag indicating if the data is null

AsBoolean
AsDate
AsFloat
AsInteger
AsString
AsVariant
DataType
Name
Null

CAstaParamItem

```
Property AsBoolean As Boolean   Member of AstaCOMClient.CAstaParamitem   Property AsBoolean
```

Description

Retrieves the Boolean value of an CAstaParamItem in Result:Boolean.

CAstaParamItem

```
Property AsDate As Date    Member of AstaCOMClient.CAstaParamitem   Property AsDate
```

Description

Sets an CAstaParamItem at Index to a DateType Value with ParamName Name.

CAstaParamItem

```
Property AsFloat As Double    Member of AstaCOMClient.CAstaParamitem   Property AsFloat
```

Description

Sets a Param Item at Index to to a float value.

CAstaParamItem

```
Property AsInteger As Long    Member of AstaCOMClient.CAstaParamitem   Property AsInteger
```

Description
Sets an CAstaParamItem at Index to LongInt with a ParamName of Name.

Example

```
AstaPlSetLongInt(RefLib,Params,1,45,'Age');
```

CAstaParamItem

```
Property AsString As String    Member of AstaCOMClient.CAstaParamitem   Property AsString
```

Description

Sets a String CAstaParamItem

CAstaParamItem

AstaFieldType=(pftUnknown, pftByte, pftSmallint, pftLongint, pftBoolean, pftSingle, pftDouble, pftDate, pftTime, pftDateTime, pftString, pftBlob);
Description

Value	Meaning
pftUnknown	
pftByte	
pftSmallint	
pftLongint	
pftBoolean	
pftSingle	
pftDouble	
pftDate	
pftTime	
pftDateTime	
pftString	
pftBlob	

CAstaParamItem

```
Property AsVariant As Variant    Member of AstaCOMClient.CAstaParamitem    Property AsVariant
```

Description
Sets or gets a Variant

CAstaParamItem

```
Property DataType As Integer    Read-only    Member of AstaCOMClient.CAstaParamitem
Property DataType
```

Description

Retrieves a AstaFieldtype value of an CAstaParamItem in Result.

CAstaParamItem

```
Property Name As String    Member of AstaCOMClient.CAstaParamitem    Property Name
```

Description

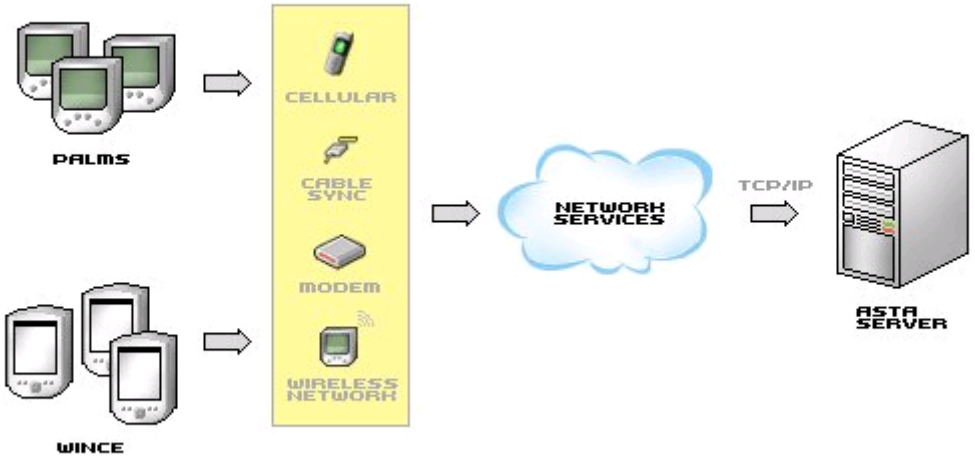
Sets an CAstaParamItem Name to Value

CAstaParamItem

```
Property Null As Long    Member of AstaCOMClient.CAstaParamitem    Property Null
```

Description
Sets an CAstaParamItem to Null.

1.10.2.4.3.7 Visual C++



The ASTA SkyWire architecture consists of remote thin clients, connecting to a middle tier Application Server via TCP/IP. SkyWire Servers are available using a number of languages include Visual Basic, Delphi and Java. This document describes how to use Visual C++ Clients to access remote SkyWire Servers servers and perform standard tasks using the SkyWire component.

ASTA Skywire consists of a middleware Server and client side components to access that server. The following components are used by SkyWire clients to perform standard tasks.

AstaConnection
AstaParamList
AstaParamValue
AstaDataList
AstaNestedParamList

The AstaClientConnection is the client end of the TCP/IP connection. It connects the ASTA client to the ASTA SkyWire Server. The Address and Port properties on the ASTA client must match the Address and Port properties on the ASTA SkyWire Server. Typically the handheld device would have a form where the IPAddress and Port can be input by the user so that the AstaClientConnection can connect to a SkyWire server and authenticate using a Username and Password.

AstaConnClose
AstaConnCreate
AstaConnDestroy
AstaConnGetFile
AstaConnGetLastError
AstaConnGetUserError
AstaConnOpen
AstaConnReceiveParamList
AstaConnSendParamList
AstaConnSetAuthenticationOff
AstaConnSetAuthenticationOn

AstaConnSetCompressionOff
AstaConnSetCompressionOn
AstaConnSetEncryptionOff
AstaConnSetEncryptionOn
AstaConnSetHttpOn
AstaConnSetHttpOff
AstaConnSetTimeout
AstaRequestUpdate

AstaConnection

```
bool AstaClConvCloseConnection(AstaHandle Conv);
```

Description

Closes a connection opened after calling AstaConnOpen.

AstaConnection

```
AstaHandle AstaClConvCreate();
```

Description
Creates an AstaHandle to an Asta Connection component.

AstaConnection

```
Err AstaConnDestroy(UInt refNum, VoidPtr hConnInstance)
```

Description
Destroys and AstaConnection and releases it's resources that has been created with AstaConnOpen.

Example

```
AstaConnDestroy(RefLib, Conversation);
```

AstaConnection

extern bool __stdcall AstaClConvGetFile(AstaHandle Conv, ULONG BlockSize, LPTSTR RequestString, LPTSTR LocalFile);

Description
Retrieves a remote file from an ASTA Server in BlockSize "chunks". This allows, for example, for large images to be streamed down efficiently.

AstaConnection

```
DWORD  AstaClConvGetLastError(AstaHandle Conv);
```

Description
Returns the last error generated by the AstaClientConnection. AstaSky Wire servers are codedso as to return an Integer value on each client request where a non-zero value is considered to be an error.

AstaConnection

```
extern DWORD __stdcall AstaClConvGetUserError(AstaHandle Conv);
```

escription
Returns the last error generated by the AstaClientConnection. AstaSky Wire servers are coded so as to return an Integer value on each client request where a non-zero value is considered to be an error. Typically errors are also returned with the server side error message or exception in an AstaParamValue of Name "Error"

AstaConnection

```
bool __stdcall AstaClConvOpenConnection(AstaHandle Conv, LPTSTR address, Word Port);
```

Description
Attempts to connect to a remote AstaSkyWire server using the Address and Port and will return a False a connection to the server cannot be achieved. No data is transferred to the server. The Asta SkyWire shared library must already be loaded.

AstaConnection

```
extern bool __stdcall AstaClConvReceiveParamList(AstaHandle Conv, AstaHandle ParamLst);
```

Description
Receives an AstaParamList from a remote ASTA Server where the hPLIntance is an AstaParamList that has been created with a call to AstaPLCreate. If the ParamList was sucessfully retrieved from the server the Var Result:Boolean would be set to True.

AstaConnection

```
extern bool __stdcall AstaClConvSendParamList(AstaHandle Conv, ULONG MsgToken, AstaHandle
ParamLst);
```

Description
Sends an AstaParam to a remote Server. The AstaParamList must have been created and typically is filled with values before being sent to the remote server. The MsgToken:UInt32 will determine how the server processes the request. If the ParamList was successfully sent to the server, teh Var Result:Boolean will be set to true.

AstaConnection

```
Err AstaConnSetAuthorizationOff(UInt refNum, VoidPtr hConnInstance)
```

Description
Turns off any Authentication information being sent to the AstaSkyWire server on connect.

Example
```
AstaConnSetAuthorizationOff(RefLib,Conversation);
```

AstaConnection

```
extern void __stdcall AstaClConvSetAuthorization(AstaHandle Conv, bool ea, LPTSTR UserName,
LPTSTR Password, bool SecurePassword);
```

Description
Sets the Username and password to be sent to the remote ASTA SkyWire server to authenticate User Information.

AstaConnection

```
Err AstaConnSetCompressionOff(UInt refNum, VoidPtr hConnInstance)
```

Description
Sets Compression off. Requires AstaZLib.prc

Example
```
AstaConnSetCompressionOff(RefLib,Conversation);
```

AstaConnection

```
extern void __stdcall AstaClConvSetCompression(AstaHandle Conv, bool zc, int level);
```

Description
Sets Compression on where level is the compression factor and the higher the number the greater the compresssion but is also slower.

AstaConnection

```
Err AstaConnSetEncryptionOff(UInt refNum, VoidPtr hConnInstance)
```

Description
Sets AstaAES Encryption off.

Example
AstaConnSetEncryptionOff(RefLib,Connection);

AstaConnection

```
extern void __stdcall AstaClConvSetEncryption(AstaHandle Conv, bool ae, void* key, aes_key
mode);
```

Description
Sets EncryptionOn and passes in a Key that is used to encrypt and decrypt the data. The Server must be aware of this key. KeyMode signifies which direction and it should usually set to both (2).

Example
AstaConnSetEncryptionOn(`RefLib,Conversation,'MumsTheWord',2);`

AstaConnection

function AstaConnSetHttpOff(refnum: UInt16; hconnInstance: Pointer): Err

Will be available in Astalib 1.1 and necessary for ASTA Palm SOAP support along with the ASTA Palm XML Parser

Description
Turns off http tunneling that was set with AstaConnSetHttpOn.

AstaConnection

function AstaConnSetHttpOn(refnum: UInt16; hConnInstance: Pointer; Address: PChar; Port: Word; Page: PChar;
Authenticate: Boolean; UserName, Password: Pchar; UseProxy: Boolean; ProxyAddress: PChar; ProxyPort: Word; ProxyAuthenticate: Boolean; ProxyUserName: Pchar; ProxyPassWord: Pchar): Err

Will be available in Astalib 1.1 and necessary for ASTA Palm SOAP support along with the ASTA Palm XML Parser

Description
Uses http instead of tcp/ip to allow access through firewalls and through a WebServer using AstaHttp.dll.

Address:Pchar Address of the ASTA Server
Port: Word : Port of the ASTA Server
Authenticate:Boolean': Authenticate at the server
UserName:Pchar UserName
Password: Pchar; Password
UseProxy: Boolean Use a proxy Server
ProxyAddress: PChar; Address of the Proxy Server
ProxyPort: Word Port of the Proxy Server
ProxyAuthenticate: Boolean Autentication required at the Proxy
ProxyUserName: Pchar Username used by Proxy Server
ProxyPassWord: Pchar Password used by Proxy Server

Example

AstaConnection

```
extern void __stdcall AstaClConvSetAstaServerVersion(AstaHandle Conv, ULONG ServerVersion);
```

Description
Asta pda libraries can support ASTA for Windows and AstaInterOp for Linux and Windows. This swtich sets the ASTA library.

Valid values are

Asta = 2
AstaIO = 3

AstaConnection

```
void __stdcall AstaClConvSetTimeout(AstaHandle Conv, ULONG Timeout);
```

Description
Sets the value in Milliseconds that an AstaConnection will "block" in waiting for a response to the server. Since there is no threading on the Palm, sockets must connect to a server and then block or wait.

Example
AstaConnSetTimeout(reflib,Connection,5000);

AstaConnection

```
Err AstaRequestUpdate(UInt refNum, VoidPtr hConnInstance, VoidPtr Buffer, Int32 BufLen, Int32
*Result)
```

Description
As part of the ASTA Hitchhikers Guide to the Wireless Universe, SkyWire clients have the ability to update themselves from remote servers. This call requests an update from a remote server.

AstaConnection

```
extern void __stdcall AstaClConvSetTerminateConn(AstaHandle Conv, bool value);
```

Description

This will the AstaConnection to terminate the connection with the server after each connection. The default is to set this to true and is recommended over any wireless connection.

The AstaDataList is an efficient DataStructure that handles rows and columns much like a database

table. To request a DataList from a remote server you must create an AstaParamList and fill it with the following values:

AstaPLSetString(AstaLibRef, params, 0,'Select * from Customer', "PalmSelect");
AstaPLSetLongInt(AstaLibRef, params, 1, 10000, "RowCount");

Param 0: the Select SQL as a String
Param 1 : the maximum rowcount

The DataList basically contains a List of ParamLists that contain data natively, stored very efficiently, wthout duplicating paramnames, like the AstaNestedParamList does. Each row can be retrieved into a ParamList with the option of bring the Param or "FieldNames" also by calling AstaDLGetAsParamList

AstaDLCreate
AstaDLDestroy
AstaDLGetAsParamList
AstaDLGetField
AstaDLGetFieldByName
AstaDLGetFieldByNo
AstaDLGetFieldCount
AstaDLGetFieldName
AstaDLGetFieldNameC
AstaDLGetFieldNo
AstaDLGetFieldOptions
AstaDLGetFieldType
AstaDLGetIndexOfField
AstaDLGetRecordsCount
AstaDLIsEmpty

AstaDataList

```
Err AstaDLCreate(UInt refNum, VoidPtr Buffer, Int32 BufLen, VoidPtr *hDLInstance)
```

Description
A DataList is created on a remote ASTA server in response to a select SQL . PdaMultiSQLSelect, or custom result set being streamed back to an ASTA Pda client.

AstaDLCreate Creates a DataList from a paramList AstaParamValue streamed over the wire as a Blob Param.

AstaDataList

```
Err AstaDLDestroy(UInt refNum, VoidPtr hDLInstance)
```

Description
Destroys an AstaDataList and frees up it's resources;

Example
AstaDLDestroy(refLib,DataList);

AstaDataList

```
Err AstaDLGetAsParamList(UInt refNum, VoidPtr hDLInstance, unsigned long recordNumber, bool
putNames, VoidPtr *hPLInstance)
```

Description
Retrieves an AstaParamList at the row designated as RecordNumber into the Var Result:Pointer. The number of rows or records can be obtained by calling AstaDLGetRecordsCount.

Example
AstaDLGetAsParamList(refLib,DataList,1,True,TempParamList);

AstaDataList

```
Err AstaDLGetField(UInt refNum, VoidPtr hDLInstance, UInt32 Index, VoidPtr *hInstance)
```

Description
Returns a handle to a Field by a given Index.

AstaDataList

```
Err AstaDLGetFieldByName(UInt refNum, VoidPtr hDLInstance, CharPtr Name, VoidPtr *hInstance)
```

Description
Returns the "handle" to the "current" field by given name

AstaDataList

```
Err AstaDLGetFieldByNo(UInt refNum, VoidPtr hDLInstance, UInt32 Index, VoidPtr *hInstance)
```

Description
Returns the "handle" to the field by given index

AstaDataList

```
Err AstaDLGetFieldCount(UInt refNum, VoidPtr hDLInstance, long *Result)
```

Description
Returns the number of fields in a DataLis in the Var Result: UInt32.

Example
AstaDLGetFieldCount(refLib,DataList,Fieldcount);

AstaDataList

```
Err AstaDLGetFieldName(UInt refNum, VoidPtr hDLInstance, VoidPtr  hFieldInstance, CharPtr
NameBuf, Int32 *BufLen)
```

Description
Returns the fieldname of a given field when a Field Handle is passed in as well as the length of the field in BufLen

AstaDataList

```
Err AstaDLGetFieldNameC(UInt refNum, VoidPtr hDLInstance, VoidPtr hFieldInstance, CharPtr
NameBuf)
```

Description
Returns the name of the field

AstaDataList

```
Err AstaDLGetFieldNo(UInt refNum, VoidPtr hDLInstance, VoidPtr hFieldInstance, UInt32 *Result)
```

Description
Returns the Nnumber of the field.

AstaDataList

```
Err AstaDLGetFieldOptions(UInt refNum, VoidPtr hDLInstance, VoidPtr hFieldInstance, UInt32
*Result)
```

```
#define pfaHiddenCol        1
#define pfaReadonly         2
#define pfaRequired         4
#define pfaLink             8
#define pfaUnNamed          16
#define pfaFixed                3
```

Description
Returns the options of the field.

AstaDataList

```
Err AstaDLGetFieldType(UInt refNum, VoidPtr hDLInstance, VoidPtr hFieldInstance, AstaFieldType
*Result)
```

Description
Returns the type of the field for a given Field Handle

AstaDataList

```
Err AstaDLGetIndexOfField(UInt refNum, VoidPtr hDLInstance, VoidPtr hFieldInstance, UInt32
*Result)
```

Description
Returns the "index" of the field in Result of a given Field Handle.

AstaDataList

```
Err AstaDLGetRecordsCount(UInt refNum, VoidPtr hDLInstance, UInt32 *Result)
```

Description
Returns the number of records contained in an AstaDataList in Result. This count usually corresponds to the number of rows requested to be returned from a remote server.

AstaDataList

```
Err AstaDLIsEmpty(UInt refNum, VoidPtr hDLInstance, bool *Result)
```

Description

If Result then the AstaDataList is Empty.

The AstaNestedParamList provides a datatructure that works like a DataSet with a row/column abstraction in order to handle SQL result requests from ASTA SkyWire servers with all the data stored as Strings. This makes it easy to use with User Interface Controls. To handle data natively, and a bit more efficiently use DataLists. NestedParamLists are actually an AstaParamList of AstaParamLists with the first ParamList a list of Fields and the rest of the paramlists containing the actual SQL Result sets.

Nested Param List Example

AstaNPLCreate
ASTANPLDestroy
AstaNPLGetField
AstaNPLGetFieldByName
AstaNPLGetFieldCount
AstaNPLGetRecordsCount
AstaNPLIsBOF
AstaNPLisEmpty
AstaNPLIsEOF
AstaNPLNext
AstaNPLPrevious

AstaNestedParamList

```
Err AstaNPLCreate(UInt refNum, VoidPtr hPLInstance, VoidPtr *hNPLInstance)
```

Description
Creates an AstaNestedParamList in Result from an incoming ParamList from the server in response to PdaToken PdaNestedSQLSelect (1010)

Example

```
AstaNPLCreate(RefLib, ParamList, NPL);
```

AstaNestedParamList

```
Err AstaNPLDestroy(UInt refNum, VoidPtr hPLInstance)
```

Description
Disposes of an AstaNestedfParamList previously created with AstaNPLCreate and releases it's resources.

Example

AstaNPLDestroy(RefLib, NPL);

AstaNestedParamList

```
Err AstaNPLGetField(UInt refNum, VoidPtr hPLInstance, UInt32 Index, VoidPtr *hInstance)
```

Description
Retrieves a Field as an AstaParamValue from a NestedParamList.

AstaNestedParamList

```
Err AstaNPLGetFieldByName(UInt refNum, VoidPtr hPLInstance, CharPtr Name, VoidPtr *hInstance)
```

Description

Retrieves an AstaNestedParamList Field byName into Result.

AstaNestedParamList

```
Err AstaNPLGetFieldCount(UInt refNum, VoidPtr hPLInstance, long *Result)
```

Description
Returns the number of Fields in a AstaNestedParamList

AstaNestedParamList

```
Err AstaNPLGetRecordsCount(UInt refNum, VoidPtr hPLInstance, long *Result)
```

Description

Returns the number of rows in an AstaNestedParamList where each row is an AstaParamList

Example
AstaNPLGetRecordsCount(RefLib, NPL, Count);

AstaNestedParamList

```
Err AstaNPLIsBOF(UInt refNum, VoidPtr hPLInstance, bool *Result)
```

Description

Returns whether the postion of the AstaNestedParamList is Beginning of File similar to the TDataSet.BOF Call

AstaNestedParamList

```
Err AstaNPLIsEmpty(UInt refNum, VoidPtr hPLInstance, bool *Result)
```

Description
Checks to see if an AstaNestedParamList is Empty and returns a boolean in Var Result if it is.

Example

AstaNPLIsEmpty(RefLib, NPL, BoolRes);

AstaNestedParamList

```
Err AstaNPLIsEOF(UInt refNum, VoidPtr hPLInstance, bool *Result)
```

Description

Returns whether the postion of the AstaNestedParamList is End of File similar to the TDataSet.EOF Call

Example

AstaNestedParamList

```
Err AstaNPLNext(UInt refNum, VoidPtr hPLInstance, bool *Result)
```

Description

Moves to the next embedded AstaParamList from an AstaNestedParamlist

AstaNestedParamList

```
Err AstaNPLPrevious(UInt refNum, VoidPtr hPLInstance, bool *Result)
```

Description

Moves to the previous embedded AstaParamList from an AstaNestedParamlist

The ASTA Unified Messaging Initiative allows for cross platform messaging by using AstaParamLists which are easy to use and fully streamable Data containers that can be transported between ASTA clients and servers and between ASTA clients regardless of hardware or operating system.

AstaPLDestroy
AstaPLClear
AstaPLCopyTo
AstaPLCreate
AstaPLCreateTransportList
AstaPLGetCount
AstaPLGetParameter
AstaPLGetParameterByName
AstaPLGetParameterIndex
AstaPLSetBlob
AstaPLSetBoolean
AstaPLSetByte
AstaPLSetDate
AstaPLSetDateTime
AstaPLSetDouble
AstaPLSetFromTransportList
AstaPLSetLongInt
AstaPLSetNull
AstaPLSetSingle
AstaPLSetSmallInt
AstaPLSetString
AstaPLSetTime

AstaParamList

```
Err AstaPLClear(UInt refNum, VoidPtr hPLInstance)
```

Description

Clears the ParamList of all items.

Example

AstaPLClear(reflib,AstaParamList);

AstaParamList

```
Err AstaPLCopyTo(UInt refNum, VoidPtr hPLInstance, VoidPtr hDestPLInstance)
```

Description

Copes one AstaParamList to another.

AstaParamList

```
Err AstaPLCreate(UInt refNum, Int32 InitialSize, VoidPtr *hPLInstance)
```

Description

Creates an AstaParamList

AstaParamList

```
Err AstaPLCreateTransportList(UInt refNum, VoidPtr hPLInstance,
                              VoidPtr *Buffer, Int32 *BufLen)
```

Description
Used internally by AstaLIb.prc to stream an AstaParamList.

AstaParamList

```
Err AstaPLDestroy(UInt refNum, VoidPtr hPLInstance)
```

Description

Destroys an AstaParamList and releases resources.

Example

AstaParamList

```
Err AstaPLGetCount(UInt refNum, VoidPtr hPLInstance, Int32 *Result)
```

Description

Returns the number of items in the AstaParamList

Example
AstaPLGetCount(reflib,ParamList,Count);

AstaParamList

```
Err AstaPLGetParameter(UInt refNum, VoidPtr hPLInstance,
Int32 Index, VoidPtr *hParamInstance)
```

Description

Retrieves an AstaParamValue at Index from an AstaParamList

Example
var
ParamList : pointer;
ParamItem:Pointer;

AstaPlGetParameter(RefLib, ParamList, 0, ParamItem);

AstaParamValue

```
Err AstaPLGetParameterByName(UInt refNum, VoidPtr hPLInstance, CharPtr Name, VoidPtr
*hParamInstance)
```

Description

Retrieves an AstaParamValue by ParamName from an AstaParamList

Example

AstaParamValue

```
Err AstaPLGetParameterIndex(UInt refNum, VoidPtr hPLInstance,VoidPtr hParamInstance, Int32
*Result)
```

Description
Returns the Index of an AstaParamValue from an AstaParamList

AstaParamList

```
Err AstaPLSetBlob(UInt refNum, VoidPtr hPLInstance,
Int32 Index, VoidPtr value, Int32 ValueLen, char *Name)
```

Description

Sets an AstaParamValue with a buffer of Value of Length ValueLen.

Example

AstaPlSetBlob(RefLib,Params,0,TheBlob,BlobSize,'Blob');

AstaParamList

```
Err AstaPLSetBoolean(UInt refNum, VoidPtr hPLInstance,
Int32 Index, bool value, char *Name)
```

Description
Sets an AstaParamValue at Index to to a Boolean value with ParamName Name.

AstaParamList

```
Err AstaPLSetByte(UInt refNum, VoidPtr hPLInstance,
Int32 Index, char value, char *Name)
```

Description

Sets an AstaParamValue at Index to a Byte value with ParamName Name.

Example

AstaParamList

```
Err AstaPLSetDate(UInt refNum, VoidPtr hPLInstance,
Int32 Index, DateTimePtr value, char *Name)
```

Description

Sets an AstaParamValue at Index to a DateType Value with ParamName Name.

Example

AstaParamList

```
Err AstaPLSetDateTime(UInt refNum, VoidPtr hPLInstance,
Int32 Index, DateTimePtr value, char *Name)
```

Description
Sets an AstaParamValue at Index to a DateTime value with ParamName Name.

AstaParamList

```
Err AstaPLSetDouble(UInt refNum, VoidPtr hPLInstance,
Int32 Index, double value, char *Name)
```

Description
Sets a Param Item at Index to to a double value with ParamName Name.

AstaParamList

```
Err AstaPLSetFromTransportList(UInt refNum, VoidPtr hPLInstance,
                               VoidPtr Buf, Int32 BufLen)
```

Description
Used internally to receive a raw buffer and convert to an AstaParamList.

AstaParamList

```
Err AstaPLSetLongInt(UInt refNum, VoidPtr hPLInstance,
Int32 Index, long value, char *Name)
```

Description
Sets an AstaParamValue at Index to LongInt with a ParamName of Name.

Example

AstaPlSetLongInt(RefLib,Params,1,45,'Age');

AstaParamList

```
Err AstaPLSetNull(UInt refNum, VoidPtr hPLInstance,
Int32 Index, AstaFieldType fType, char *Name)
```

Description
Sets an AstaParamValue to Null.

Example
AstaPLSetNull(RefLib,Params,1,pftString,'Department');

AstaParamList

```
Err AstaPLSetSingle(UInt refNum, VoidPtr hPLInstance,
Int32 Index, float value, char *Name)
```

Description
Sets a Param Item at Index to to a Single value with ParamName Name.

Example

AstaParamList

```
Err AstaPLSetSmallInt(UInt refNum, VoidPtr hPLInstance,
Int32 Index, Int32 value, char *Name)
```

Description

Sets an AstaParamValue at Index to a SmallInt value with ParamName Name.

Example

AstaParamList

```
Err AstaPLSetString(UInt refNum, VoidPtr hPLInstance,
Int32 Index, CharPtr value, char *Name)
```

Description

Sets a String AstaParamValue at index with ParamName of Name

Example

AstaPlSetString(RefLib, ParamList, 0, 'Select * from Customers', 'PalmSelect');

AstaParamList

```
Err AstaPLSetTime(UInt refNum, VoidPtr hPLInstance,
Int32 Index, DateTimePtr value, char *Name)
```

Description

Sets a Time AstaParamValue at index with ParamName of Name

Example

AstaParamValues are the items contained in the expandable AstaParamLists explained more fully in the ASTA Unified Messaging Initiative

Parameter name - the name of the parameter
Parameter type - the mode of the parameter, in, out, inout, result.
Data type - AstaFieldType which follows the database types (char, bit, integer, etc.)
IsNull flag - a flag indicating if the data is null (a bit representing true or false)
Bytes of data - the actual data as a list of bytes

AstaPLparamGetAsBlob
AstaPLParamGetAsBoolean
AstaPLParamGetAsDate
AstaPLParamGetAsDateTime
AstaPLParamGetAsDouble
AstaPLParam.GetAsLongInt
ASTAPLParamGetAsString
AstaPLParamGetAsStringC
AstaPLParamGetAsTime
AstaPLParamGetName
AstaPLParamGetType
AstaPLParamGetIsNull
AstaPLParamSetName

AstaFieldType=(pftUnknown, pftByte, pftSmallint, pftLongint, pftBoolean, pftSingle, pftDouble, pftDate, pftTime, pftDateTime, pftString, pftBlob);

Description

Value	Meaning
pftUnknown	
pftByte	
pftSmallint	
pftLongint	
pftBoolean	
pftSingle	
pftDouble	
pftDate	
pftTime	
pftDateTime	
pftString	
pftBlob	

AstaParamValue

```
Err AstaPLParamGetAsBlob(UInt refNum, VoidPtr hInstance,
VoidPtr Buffer, Int32 *BufLen)
```

Description

Copies an AstaParamValue Data into Buffer and returns the size of the Blob in BufLen.

AstaParamValue

```
Err AstaPLParamGetAsBoolean(UInt refNum, VoidPtr hInstance, bool *Result)
```

Description

Retrieves the Boolean value of an AstaParamValue in Result:Boolean.

AstaParamValue

```
Err AstaPLParamGetAsDate(UInt refNum, VoidPtr hInstance,
DateTimePtr Date, bool *Succeeded)
```

Description

Retrieves the DateTimePtr value of an AstaParamValue in Date and if successful returns True in Result.

AstaParamValue

```
Err AstaPLParamGetAsDateTime(UInt refNum, VoidPtr hInstance, DateTimePtr Time, bool
*Succeeded)
```

Description

Retrieves the DateTimePtr value of an AstaParamValue in Time and if successful returns True in Result.

AstaParamValue

```
Err AstaPLParamGetAsDouble(UInt refNum, VoidPtr hInstance,
double *Result)
```

Description
Retrieves a double value of an AstaParamValue in Result.

AstaParamValue

```
Err AstaPLParamGetAsLongInt(UInt refNum, VoidPtr Instance, long *Result)
```

Description

Retrieves a LongInt value of an AstaParamValue in Result.

AstaParamValue

```
Err AstaPLParamGetAsString(UInt refNum, VoidPtr hInstance,
CharPtr NameBuf, Int32 *BufLen)
```

Description
Retrieves the AstaParamValue into NameBuf and the length of the Pchar into BufLen.

AstaParamValue

```
Err AstaPLParamGetAsStringC(UInt refNum, VoidPtr hInstance, CharPtr NameBuf)
```

Description
Retrieves the AstaParamValue into NameBuf

AstaParamValue

```
Err AstaPLParamGetAsTime(UInt refNum, VoidPtr hInstance,
DateTimePtr Time, bool *Succeeded)
```

Description

AstaParamValue

```
Err AstaPLParamGetIsNull(UInt refNum, VoidPtr hInstance,
bool *IsNull)
```

Description
Returns True to Result of the if the ParamValue is null.

AstaParamValue

```
Err AstaPLParamGetName(UInt refNum, VoidPtr hInstance,
CharPtr NameBuf, Int32 *BufLen)
```

Description
Retrieves the Name of the ParamValue along with the buffer length.

AstaParamValue

```
Err AstaPLParamGetNameC(UInt refNum, VoidPtr hInstance,
CharPtr NameBuf)
```

Description
Retrieves the Name of the ParamValue.

AstaParamValue

```
Err AstaPLParamGetType(UInt refNum, VoidPtr hInstance,
AstaFieldType *Type)
```

Description

Retrieves a AstaFieldtype value of an AstaParamValue in Result.

AstaParamValue

```
Err AstaPLParamSetName(UInt refNum, VoidPtr hInstance, CharPtr Value)
```

Description

Sets an AstaParamValue Name to Value

1.10.3 Contact

Since 1998 has has been ASTA providing technology to allow developers to create applications that run securely over any network including the internet.

Visit the ASTA Delphi site at www.astatech.com
Visit the ASTA Wireless site at www.astawireless.com

ASTA Technology Group Inc
PO Box 425
Boise ID 83701
USA
Phone: +1 208.342.7800
Fax: +1 208.389.4643
email: info@astatech.com


ASTA
ASTA SkyWire Server
Copyright © 2001 ASTA Technology Group
ASTA
TECHNOLOGY GROUP
BRING ROCKET POWER TO YOUR APPLICATIONS

Index

- A -

- B -

- C -

- D -

- E -

- F -

- G -

- H -

- I -

- K -

- L -

- M -

- N -

- O -

- P -

- R -

- S -

- T -

- U -

- V -

- W -

- X -