

ASTA TECHNOLOGY GROUP

---

ASTA Documentation Series

# ASTA *Vision*

# ASTA Documentation Series

## ASTA Vision

---

© ASTA Technology Group  
3132 E Stone Point Drive • Boise • Idaho • 83712 • USA  
Phone +1 (208) 389.9546 • Fax +1 (208) 389.4643  
info@astatech.com

ASTA UK  
The Chapter House • Brunswick St • Whitby • YO21 1RB • UK  
Phone +44 (1947) 606 556 • Fax +44 (1947) 603 146  
info@astatech.co.uk

Document Version: 0.2 (24 October 2000)  
Date Printed: 25 October, 2000

Updated versions of this document may be found at:  
<http://www.astatech.com>  
<http://www.astatech.co.uk>

This document contains proprietary information about the ASTA development tools. All information in this document is to be treated as commercial in confidence. Under no circumstances may this document be reproduced, copied in whole or in part, or quoted without the express written permission of the authors.

ASTA Technology Group may have trademarks, patents and / or pending applications covering subject matter in this and any other accompanying documentation. The furnishing of this document does not give you any license to these patents and trademarks.

Copyright © 1997 – 2000 ASTA Technology Group. All rights reserved. Other products mentioned in this document are trademarks or registered trademarks of their respective holders.

## Table of Contents

---

<b>Preface .....</b>	<b>3</b>
Purpose of this Document .....	3

---

<b>Overview .....</b>	<b>5</b>
What is n-Tier? .....	5
The ASTA Vision .....	7
Some Key Features Of ASTA .....	8

---

<b>ASTA Versus the Browser Application .....</b>	<b>13</b>
What's Wrong With The Web? .....	14
Web Burn .....	14
Why ASTA Succeeds Where The Web Fails .....	16

---

<b>Strategies and Issues for Application Design .....</b>	<b>19</b>
The Landscape .....	19
The Golden Rule .....	20
Design Strategies .....	20
Model 1: Client Centric Design (SQL Clients) .....	20
Model 2: Server Centric Design (non-SQL Clients) .....	22
Model 3: Asynchronous Messaging .....	23
Design Issues .....	24
Asynchronous Query Support .....	24
Transaction Support .....	24
ASTA and Sockets .....	25
Database Issues .....	26
ASTA and Firewalls .....	26

---

<b>Appendix 1 Glossary .....</b>	<b>28</b>
----------------------------------	-----------

# Preface

*The purpose of this document and it's role within the ASTA Documentation Series.*

## **Purpose of this Document**

---

**T**his document provides an overview of ASTA – both in terms of it's major functionality and it's relationship to current and emerging technologies.

The document has a dual purpose. Firstly, it provides a comprehensive introduction to the ASTA technology and outlines the paradigm for developers who will be using ASTA in their day to day work. Secondly, it provides a conceptual overview for managers, designers and others who need to understand the technology, but not necessarily the mechanics of how ASTA works.

A separate ASTA Developer's Guide is available which provides detailed technical information about using ASTA in your applications. The ASTA Developer's Guide also provides tutorials and other information designed to get developers up and running with ASTA.

It is recommended that developers who will be using ASTA read both this document and the ASTA Developer's Guide to gain a full understanding of the potential uses for ASTA. To aid this process, the basic structure of both documents is the same wherever possible.

A glossary is provided at Appendix A which helps to define key terminology used throughout this document. In particular, non-developers should review the glossary before attempting to read the main body of this document as it defines some common developer terminology.

For updates to this documentation, check the ASTA websites at <http://www.astatech.com> (USA) and <http://www.astatech.co.uk> (UK).



# Overview

*An overview of ASTA and our vision.*

**A**STA 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.

## What is 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.

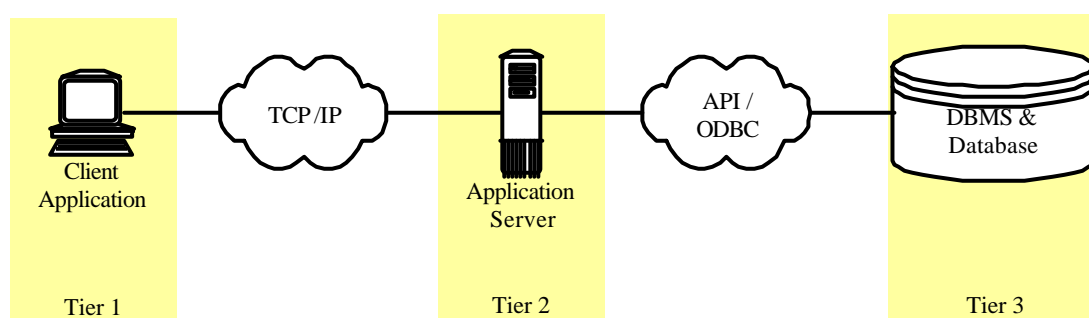
The precursor to n-tier is the client/server application where there is a delineation between the *client* (as seen by the user) and the *database server* located probably on a server machine on a network. These applications are commonly called "fat client", as most of the business logic for the application lies on the client side.

The evolution of n-tier was spurred 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*); and
- the data is located within a database which is managed by a DBMS.



**Figure 1: 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 its 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*.

## The ASTA Vision

---

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.

## **Some Key Features Of ASTA**

---

Two key benefits of ASTA's design are that:

- it abstracts the database application development process so that any database components can be used on ASTA Servers; and
- it handles all threading internally so that developers need not be concerned with threading issues.

This and other features are discussed below.

### **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.

### **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.

### **Scalability**

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.

### **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.

### **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.

### **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.

### **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.

## **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).

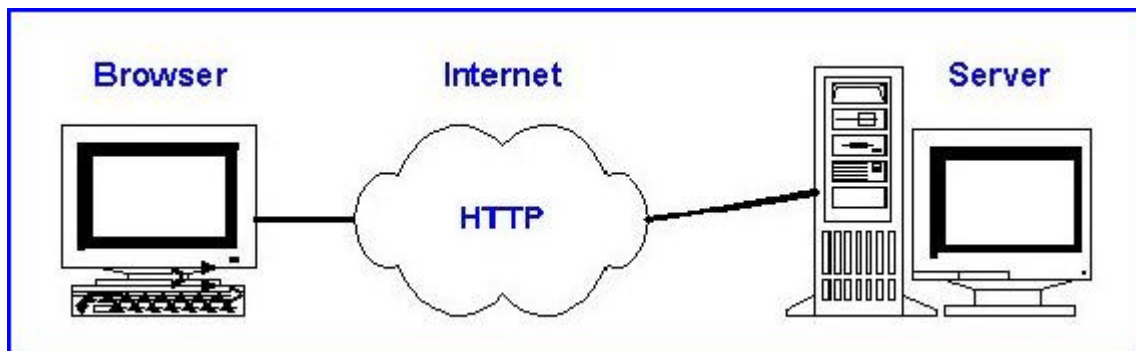


# ASTA Versus the Browser Application

*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.

## **What's Wrong With The Web?**

The problem with the web is fundamental to its architecture and manifests itself in two places, the protocol and the interface:

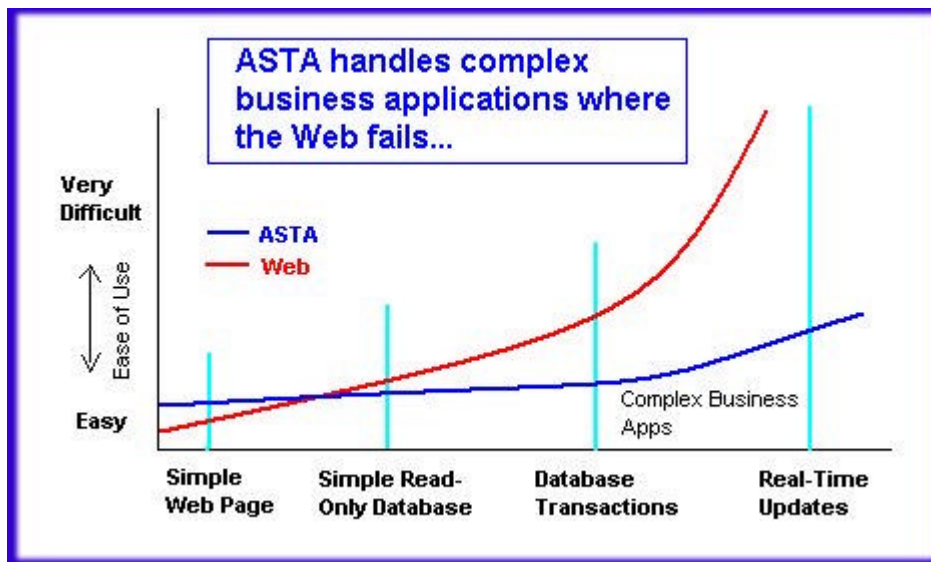
- HTTP, the language between servers and browsers, is a *stateless protocol*
- HTML has a limited interface and limited utility

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.

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.

## **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 its 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 its 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.

## **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**

As depicted in Figure 3, 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.

### **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.

## **In Summary**

ASTA is a hybrid technology that takes the traditional strengths of distributed computing and combines it with the strengths of the WWW. Programs developed with ASTA are LAN, WAN and Internet-ready. They have small footprints - typically around 1 MB - and were designed to lower administrative costs (development costs and long term system administration costs). ASTA client applications can run on inexpensive PCs, or even on handheld and other devices, can call the complete WIN32 API, and have a rich interface. ASTA allows you to design and develop programs that can change as quickly as you do. Instead of fighting implementation details and large scale system administration problems, ASTA allows you to deliver programs that are relevant to your business: programs that promote your competitiveness.

# Strategies and Issues for Application Design

*Discussion of some specific issues in the design of ASTA applications.*

## The Landscape

---

ASTA is based on the 3 Tier programming model. These tiers consist of the database, an embedded application server (the middle tier) and a client application. Simply put, you must start an ASTA server for an ASTA application to function and an ASTA client must be able to connect to that server via TCP/IP. For database access, the ASTA server must be able to connect to a database using the appropriate DBMS methods.

ASTA supports a wide variety of multi-tier development models – without forcing any of them. It allow designs which put extensive computation in the client application, and also allow extremely “thin” designs which put very little on the client and most intelligence in the server. ASTA facilitates traditional “RPC” style request-reply communications as well as “Message Oriented Middleware” style asynchronous messaging. ASTA client and server infrastructure can handle extensive database interactions automatically, or it can get out of the way and let you code all SQL yourself, on the client or server side.

The various architectures are described below as different models for application design; however, it is important to keep in mind that ASTA does not force the developer to select one model and use it pervasively; rather, the models can be mixed as needed for different aspects of the same application. You will commonly find ASTA applications built using a combination of client and server side techniques.

## The Golden Rule

---

Before going any further, it is important that you understand the one golden rule of n-tier design. This rule deserves it's own section – if not an entire book!

*When developing for n-tier, always transfer only the data you need.*

The biggest single performance issue for an n-tier application is the time taken to transfer data across a network – whether that be a LAN, WAN or the internet.

No user can absorb a grid filled with 60,000 records. If you need the user to be able to summarize the data in 3 completely unrelated ways, it is much more efficient to have 3 queries available that will gather the summarized data from the database than to bring the data across and summarize it on the client. In the n-tier world there are no excuses for transmitting huge chunks of raw data. As a developer or designer of n-tier systems, a guiding principle must always be to devise ways of minimizing network traffic.

## Design Strategies

---

### ***Model 1: Client Centric Design (SQL Clients)***

ASTA supports pure n-tier programming, but also supports client side SQL. Using our embedded application servers, client side SQL allows you to write client database applications that can run over the internet without coding the middle tier. Think of ASTA client side SQL as “fat client programming” but without any installation issues and little or no learning curve if you already use SQL. No matter what database access methods you require (eg BDE, ADO, Sybase or Oracle client DLLs) their resources will only need to be installed in one place – on the middle tier for use by the application server. ASTA clients require no DLL's, setup or registration. They just need to be able to connect to an ASTA server somewhere.

ASTA was designed so that Delphi database developers could use their existing skills. This means that if you know normal dataset component methods like Append, Edit and Post you will be able to use ASTA very easily. With ASTA, your developers only need to write SQL to select the data your client application needs. Our theory is that you should be able to select the data you want, let the user modify it as appropriate, and then when you are ready, just post your changes to the server and let ASTA handle all of the update logic automatically

for you. ASTA's suitcase model of course supports all of this even if you disconnect from the server and are offline for a week.

One of the first concepts you will have to understand is that we don't want to go to the server too often. ASTA provides much better performance than client server drivers and components because network communication is kept to a bare minimum. Performance is always tied to how often you go to the server and how large a result set you request from the server.

ASTA uses very fast in-memory datasets on the client. In fact the ASTA dataset component is so fast and easy to use that it is used internally by ASTA to store disparate data. There is an ASTA dataset that is a pure in-memory dataset and then there is a descendent of this dataset designed for use on the client (referred to as the client dataset). Therefore, when a client sends a request for data to the server, the results are streamed back to the client and held in memory in a client dataset. This means that there are no open tables on the server, and no locked rows.

When you write client side SQL and then activate the client dataset, the SQL statement is sent to the server along with other internal information (eg whether or not you want to bring back memos and blobs). The server executes the query, packs up the data, and streams the results back to the client.

When a normal 2 tier application executes a query it does not iterate through the whole result set – it just runs the query and presents you with the results. However, when you do a select from an ASTA server there are additional steps that must be executed by ASTA before the data can be used by the client.

1. The query executes. If you are going to do any performance testing with ASTA versus a 2 tier application, this is the piece the 2 tier application does.
2. After the query executes, ASTA servers must iterate through all the rows and pack up the data. In large scale databases just getting the number of rows can be an expensive proposition. For instance, with many DBMSs, if you request a record count a separate query is run in order to determine the number of records in the relevant table.
3. After the dataset is packed up it must be transported via TCP/IP. TCP/IP transports data in 4K (4096 bytes) chunks. Therefore, even very small result sets will cost the same as a 4K fetch.
4. When the data packet arrives on the client, it must be unpacked into the client dataset.

ASTA provides a number of options for managing the synchronization of data between client and server. However, we recommend that you use the Cached Edit Mode. This means that after you have selected

your data from the server, you can edit, insert, delete on the client as long as you want and then post the changes to the server with a single call. You can even disconnect from the server, save the dataset to file, wait a week, do more edits, inserts and deletes, connect back to the server and apply the updates. ASTA automatically generates the required SQL to apply your updates as a single database transaction so it is very efficient.

Alternately, the option of immediately posting individual changes is provided where this is necessary. An SQL statement is generated automatically by ASTA after each individual edit and sent to the server to be executed. Performance will be very fast but in the same trip to the server you could have also sent many rows to populate the 4K packet that TCP/IP sends.

## ***Model 2: Server Centric Design (non-SQL Clients)***

### ***Server Side Coding***

When discussing “n-tier” technology the phrase “allows you to put the business logic on the middle tier” is often heard. Normal 2 tier applications use SQL on the client, and this technique has been described above as used with ASTA to create internet enabled client server applications. But there are occasions where for security, maintainability, code reuse or performance you may want to move that SQL to the server. ASTA fully supports this with providers, business objects and messaging. What follows is an overview of what is possible using ASTA and server side n-tier coding.

### ***Server Side Processing with Business Objects***

Let's say you have a complicated report that needs to be run weekly that uses a lot of queries. Instead of writing SQL on the client, with ASTA you can use a business objects manager to write a server side method that will trigger code you can write on the server. The client dataset can see a list of all of available server methods. Therefore, instead of opening up a number of datasets on the client, a very small message will be sent to the server that triggers code that is tied to the server method. All of the queries are executed on the server and their results are combined into a single in-memory dataset and streamed down to the client. All the processing and fetching took place on the server so the resultant network traffic was kept to a minimum.

### ***Server Side SQL using Providers***

If you want to keep your SQL on the server and not have any SQL on the client, you can use ASTA Providers. Providers can basically hook into any dataset components, allowing SQL to be written on the server using any dataset components you desire. As with business objects above, ASTA client datasets can see all providers on the server.

Instead of writing SQL on the client side, the client chooses the appropriate provider available on the ASTA server. When you choose a provider and activate the client dataset, a small message is sent to the server to tell the provider to trigger the connected dataset to get its data from the database. Any result sets are streamed back to the client. If the result set needs to be updateable, then you simply set a couple of properties of the provider. On the client, you tell the client dataset to generate all SQL on the server. Now you can append, edit and delete just like with Cached Edit Mode in the client side SQL option (described in the previous section). The only difference is that this time the server generates the SQL to provide the updates.

Using providers also allows you to broadcast any changes made by one client to any other clients who may be connected to the server.

### ***Model 3: Asynchronous Messaging***

One of the design goals of ASTA was to allow database application developers to communicate via TCP/IP without having to know anything about sockets or TCP/IP. So ASTA introduced a very easy to use messaging layer designed to shield the developer from low level socket threading issues.

ASTA messaging allows any kind of data to be easily transmitted between client and server, clients and other clients and between servers. Since TCP/IP maintains state, ASTA servers can push out messages allowing sophisticated applications such as Auction Applications to be created easily with ASTA messaging.

ASTA future plans include messaging that can communicate with disconnected users using industry standard Message Queue concepts implementing peer to peer and point to point abilities.

ASTA users have extended their servers to update themselves, do asynchronous database queries and remote control remote servers all using simple ASTA messaging calls. The ASTA auto client update feature uses ASTA messaging to perform a version check when client applications connect and stream down a more current version if one exists.

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 ASTA Parameter List. This is just like a TParamList that Delphi developers are already familiar with, but it is fully streamable. It also can contain datasets, any kind of binaries and even other parameter lists. This parameter list model has been extended to create Java, Palm and WinCE versions of the ASTA parameter lists. This allows ASTA servers to communicate parameter lists across machines on different platforms and automatically handle any reformatting of the message for non-Intel clients like those that may run for Java, the PalmOS and WinCE devices. In short, our parameter lists allow you

to send any type of information between the different tiers of your application.

Using ASTA it is also possible to send an ASTA parameter list synchronously – forcing the sender to wait until the server sends back a response. This allows developers to write procedural code rather than event driven code.

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 from servers, to sending faxes.

Once you have a middle tier you opens up a complete world of new possibilities.

## **Design Issues**

---

### ***Asynchronous Query Support***

The first two models generally describe synchronous communication between ASTA clients and servers – i.e., when using client datasets to retrieve data from the server, ASTA puts the client into a wait state whilst waiting for the server to return the result set. However, for asynchronous operation ASTA allows a remote client to send a message to the server with appropriate parameters. This allows the client to carry on doing other tasks whilst the server prepares the results. The server handles query execution and sends the results back to the client as a coded message.

This means that the client can send multiple asynchronous requests to the server, each of which will run in it's own thread and each with it's own database session. Each request will finish executing at different times depending on the size of the result set and will be streamed back to the client. The client application can then deal with the result sets as appropriate.

With power comes responsibility! As each request is launched within it's own session, it will consume resources on the server for the duration of the request, so this feature should be used sparingly.

### ***Transaction Support***

Transactions in ASTA are optimized to keep network traffic to a minimum. Starting a transaction on the client, executing some SQL statements and then ending the transaction would need many round trips to and from the server to perform each step. Therefore, within

ASTA, a complete transaction is always sent to the server to be started and ended entirely on the server side.

A complete transaction is sent to the server when you call the client side method to apply updates. Therefore, when designing your application you need to balance frequency of updates between fulfilling your concurrency requirements (ie how many connected users will you have, and how up to date does the data need to be) versus the need to minimize network traffic.

In normal 2-tier client/server programming, an application would normally start a transaction, execute some SQL and then either commit or roll back the transaction depending on whether the executed SQL succeeded or not. To facilitate efficient communication over latent connections, ASTA was designed to communicate with remote servers by executing the complete transaction on the server for you. Client/server drivers are slow or “chatty” because they make too many round trips across the network. With ASTA you only need to call the apply updates method and internally ASTA will transmit all the information needed to be used by the remote server for you. On the server, ASTA will provide a database connection to be used, start a transaction with that connection, attempt to post all the SQL and then appropriately commit or rollback the transaction. If the transaction was a success, a message will be returned to the client application so that the client can flush its pending changes. If the transaction was not a success, a message will be sent from the server which will raise an exception on the client. The client can then decide whether to cancel or revert any changes, or prompt the user to make any changes needed to resolve the problem that triggered the exception. The end result is just one message to the server and one message back from the server.

To recap, *with ASTA you only write SQL for select statements, set your edit mode, and then use normal dataset methods to append, edit and delete data within the client application.* One line of code can be used to post your changes to the server and you have a fully enabled thin client database application that can run efficiently over the internet.

### **ASTA and Sockets**

The asynchronous nature of sockets, and knowing when you can open datasets are important concepts and techniques and the faster you understand the relationship of your remote application and it's ability to communicate with an ASTA Server the easier it will be for you to be productive. Depending on your connection, whether on a LAN or over a slow dial up internet account, the socket performance may vary wildly but once you get the basics of the way things flow you will find that you can write robust and brisk ASTA client applications.

Again, remember the golden rule: always ensure that you only fetch the minimum amount data you need.

## ***Database Issues***

The ASTA team has abstracted the complete database application process so that developers can use the standard methods and behaviour they are accustomed to - no matter what database is being used on the server. This abstraction process also can allow you to more readily support different databases by allowing your ASTA client application to remain the same, but allowing you to run it against different databases or a different ASTA Server using other 3rd party Delphi database components.

ASTA provides specific support for a number of features that are traditionally available to desktop database developers but often not available (or not as reliable) in a multi-tier environment - for example, auto-increment and default value fields. ASTA deals with the details so you can use the features without undue effort.

When you insert a value into a table there may be auto-increment fields or other triggers or default values that are created by your database. Oftentimes dealing with these values created on the server is problematic in an n-tier environment. As an in-memory dataset, calling refresh after an insert on a client dataset does nothing but refresh the internal data buffers. It is normal n-tier practice to close and then re-open a client-side dataset to "refresh" the database but of course there is a performance hit if the complete result set needs to be refetched from the server.

ASTA client datasets have the ability to refetch individual field values on inserts and updates to make this process very easy. Refetches are only supported in Cached Edit Mode, and a transaction is required on the server.

### ***Sorting Data Using ASTA Data Sets***

ASTA datasets contain no indexes. Despite this, searches on the data are extremely fast because the data is held in memory. If you need to re-order your result set, you can use the sort methods of the client dataset rather than going back to the server for more data.

## ***ASTA and Firewalls***

ASTA servers require a static IP Address and a "port" to run on. There are 65,535 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 25. You can run as many ASTA servers as your hardware allows on any one machine as long as each one runs on a different port.

However, firewalls are designed to block access to a machine 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 has a number of techniques that you can use in your application to work through a firewall:

- Have the Firewall Administrator open the port your ASTA server is running on;
- Run your ASTA server on port 80 or port 8080;
- Set the ASTA server and client to format all messages as HTTP;
- Configure the ASTA client application to run stateless HTTP;
- Use ASTA’s ISAPI DLL with IIS to allow your ASTA client application to appear as a browser application;
- Use an external Proxy Server;
- Use ASTA SOCKS support; or
- Use the ASTAProxyServer.

Details of each of these options are discussed in the ASTA Developer’s Guide.

# Appendix 1

## Glossary

The following list is an alphabetically ordered set of definitions for key terms. In some cases, these definitions are simply provided in order to clarify our interpretation of terms which are commonly used. This is necessary as there are often conflicting interpretations.

Within the definitions, terms which are defined elsewhere in the glossary are highlighted in bold.

<b>application server</b>	Refer to <b>middle tier</b> .
<b>ASTA server</b>	This is the <b>middle tier</b> in a 3-tiered ASTA application. This program will usually contain a <b>server socket</b> which allows ASTA <b>client applications</b> to connect to it.
<b>asynchronous communication</b>	<p>Communication between the various parts of a multi-tier application can be either <i>synchronous</i> or <i>asynchronous</i>.</p> <p>With asynchronous communication, a <b>process</b> sends a request to an external process as a message, but does not wait to find out the results of the request. If any results are generated, they will usually be sent back to the original calling process as a separate message.</p> <p>This usually allows an application to continue working whilst the external process is preparing any results.</p> <p>Refer also to <b>synchronous communication</b>.</p>
<b>cached edit mode</b>	<p>This refers to the way ASTA reconciles changes to data made by a client application, and how such changes are propagated to the database.</p> <p>In cached edit mode, the client can retrieve some data from the database, make any changes to the data (including editing, deleting or updating the data) and then ASTA will simply compile all changes into a single <b>transaction</b> which will be sent to the server for writing to the database.</p> <p>This mode provides the most efficient use of bandwidth as it minimises the amount of network traffic generated by an ASTA application.</p>
<b>client application</b>	<p>This is the program(s) that will be run on each user's machine, and will provide the functionality necessary to connect to an ASTA server.</p> <p>It can best be thought of as the user interface into your application.</p>

<b>client dataset</b>	<p>A <b>dataset component</b> which is generally contains data within the client application.</p> <p>In an ASTA <b>client application</b>, the client dataset contains a copy of data that has been extracted from a database. The client can edit the data in this copy without affecting the actual database. If appropriate, the client application can then write any changes made in the client dataset to the database (refer to <b>cached edit mode</b> for a discussion of this).</p>
<b>client socket</b>	<p>The client side version of a <b>socket component</b>. It handles all of the client side tasks associated with allowing an ASTA client application to communicate with an <b>ASTA server</b>.</p>
<b>components</b>	<p>Within the context of this document, we are usually referring to encapsulated pieces of program code which are used by developers in the construction of their applications. For example, a button on a form will usually be created from a button component.</p> <p>Delphi and C++ Builder developers have available to them a library of both visual and non-visual components known as the VCL.</p> <p>Some of the most common components of interest to ASTA developers are database components (ie non-visual components which encapsulate the functionality required to connect to and manipulate data within various <b>DBMSs</b>) and data-aware controls (visual components which allow data from a database to be viewed and edited by the user within a client application – examples of this are a data edit box and a data grid).</p> <p>ASTA itself is provided as a set of (mainly non-visual) components.</p>
<b>database transaction</b>	<p>Refer to <b>transaction</b>.</p>
<b>dataset components</b>	<p>Within Delphi, this is a <b>component</b> which holds data which has been retrieved from a database. The data held will usually be the result of an SQL query that has been run against the database, or in the case on non-SQL dataset components, the data from a table in the database to which the component refers.</p>
<b>DBMS</b>	<p><b>DataBase Management System</b>. This refers to the processes (eg executable applications, dlls, driver libraries, etc) which are designed to manage and provide access to the data within a database.</p>
<b>middle tier</b>	<p>This is the tier which provides the main business logic of a <b>multi-tier</b> application. It allows the abstraction of the program logic within a client application so that the client has no dependencies on any specific <b>DBMS</b>.</p> <p>Within 3-tier applications, it is also called the <i>application server</i>. ASTA refers to this as the <b>ASTA server</b>.</p>
<b>multi-tier application</b>	<p>An application made up of a number of discrete processes which may be run on different physical machines. Typically, an ASTA application will be three tiered: with the tiers being (1) the <b>client application</b>, (2) the <b>ASTA server</b> and (3) the database server (or <b>DBMS</b>).</p>
<b>n-tier application</b>	<p>See <b>multi-tier</b>. The n in n-tier simply means an undefined number of tiers.</p>
<b>process</b>	<p>Within the context of this document, a process is usually either an executable program, or a library (such as a Windows DLL or an ActiveX).</p>
<b>server socket</b>	<p>The server side version of a <b>socket component</b>. It handles all of the server side tasks associated with allowing an ASTA client application to communicate with an <b>ASTA server</b>.</p>

---

<b>socket component</b>	<p>ASTA applications will usually contain at least 2 socket <b>components</b>: a <b>client socket</b> and a <b>server socket</b>. The purpose of these components is to enable the different tiers of the application to communicate with each other.</p> <p>A prime function of ASTA is to assist developers by hiding the complexities of socket communication, thus most of the hard work is done automatically by the socket components.</p>
<b>synchronous communication</b>	<p>Communication between the various parts of a multi-tier application can be either <i>synchronous</i> or <i>asynchronous</i>.</p> <p>With synchronous communication, a <b>process</b> sends a request to an external process and then 'waits' until the external process returns a result. This is the primary method of communication for ASTA client and server applications.</p> <p>Refer also to <b>asynchronous communication</b>.</p>
<b>transaction</b>	<p>Also referred to as a "database transaction", it refers to the practice of wrapping up a number of proposed changes to a database into a single task, thus helping to ensure the integrity of the underlying data. The idea being that if one or more steps should fail for any reason then the application has the opportunity to correct the failed change or undo all of the steps.</p> <p>Although by no means the rule, it is not uncommon for a failed step within a transaction to cause the whole transaction to be 'rolled back' – ie the DBMS effectively ignores all of the steps.</p>
<b>VCL</b>	<p>Visual Component Library. A set of visual and non-visual <b>components</b> which encapsulate one or more particular functionalities. Used by Delphi and C++ Builder developers in the creation of applications.</p>

---