

# **AptrixConnect White Paper**

**July 2000**

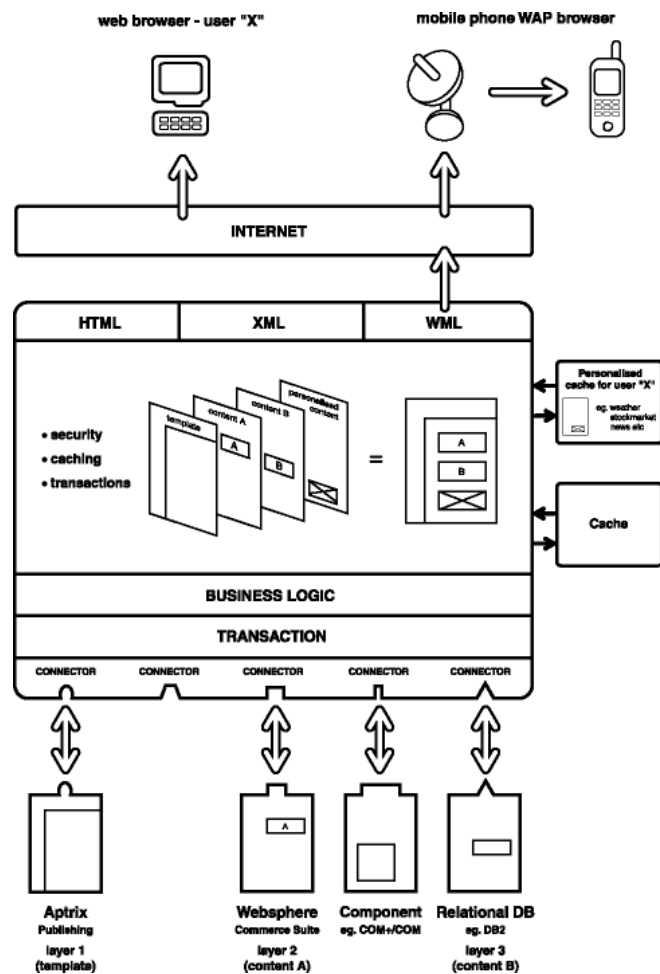
# Table of Contents

Table of Contents .....	2
Abstract .....	3
User Scenarios .....	4
Scenario 1: Merging HTML from Multiple Web Servers.....	4
Scenario 2: Interfacing To Multiple Backends.....	4
Scenario 3: Reusing Legacy Components .....	4
Using ApatrixConnect .....	5
Calling ApatrixConnect.....	5
Interfaces.....	5
For example: .....	5
Caching & Performance .....	6
Overview.....	6
Site Cache.....	6
User Caches .....	6
Cache Strategies & Control.....	6
When to Cache: The CACHE Flag.....	6
For example: .....	6
When Cache Objects Are Refreshed: The EXPIRES Attribute .....	7
For example: .....	7
Cache Control through HTTP Headers.....	7
Merging Content.....	8
ApatrixConnect: A Smart Proxy.....	8
Retrieving Content.....	8
Maintaining Links In Returned Content.....	8
For example: .....	8

# Abstract

AprixConnect is a servlet-based solution that provides superior connector functionality for integrating web applications and eBusiness solutions. It will serve as an intermediary between various servers combining and massaging output to a client in a timely manner. It is highly flexible with the ability to connect to virtually any combination of legacy backend systems and legacy components on any number of platforms and return content via multiple delivery mechanisms.

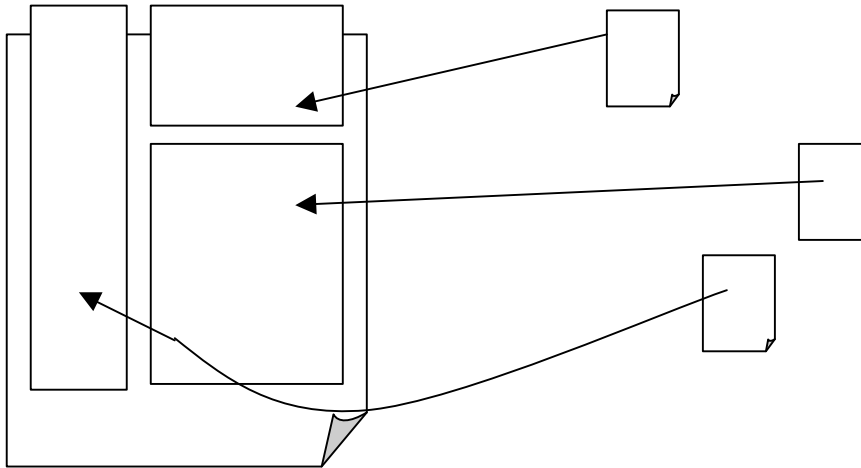
1. **Combiner** – merging data from any back end system. For example, combining Aprix with a WebSphere Commerce solution and with DB2 data to give a single cohesive yet loosely coupled solution. Marketing content can be bound with transactional content.
2. **Caching** - the ability to cache in memory whole pages or areas of pages on a per site level or a per user level.
3. **Performance** - the caching and layering in AprixConnect can be used to dramatically improve performance of a site allowing components on a page to be cached or not cached.
4. **Security Management** – AprixConnect can provide a single end point to multiple systems – including the management of the login facilities across different server environments and web sites.



# User Scenarios

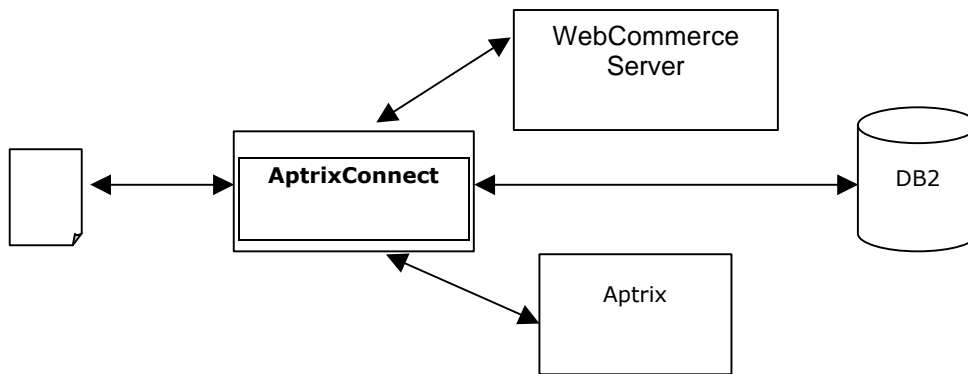
## Scenario 1: Merging HTML from Multiple Web Servers

ABank has three web servers:



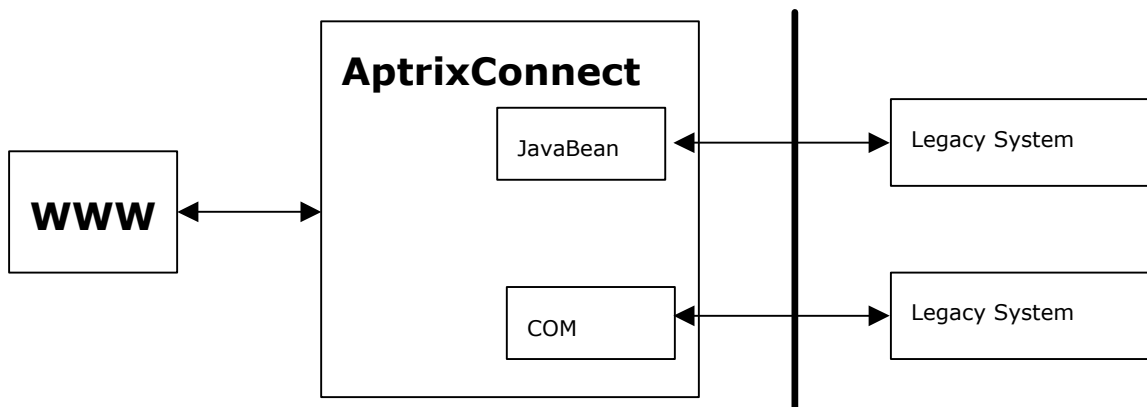
## Scenario 2: Interfacing To Multiple Backends

Empire would like to sell its office stationary products over the web. They have chosen WebSphere Commerce to handle e-commerce payments via their website but use DB2 to maintain their product catalogue and inventory. For security they have placed this product database behind a firewall.



## Scenario 3: Reusing Legacy Components

MoreMoney Accountancy has developed a number of COM and JavaBean components to interface to their legacy ledger system. They would like to allow customers a secure way of viewing tax information remotely via a browser. They would like a secure interface to their ledger system and would like to reuse these components.



# Using AprixConnect

## Calling AprixConnect

AprixConnect has been designed to act as a gateway between the outside world and an internal intranet or other servers. Accessing the gateway through a browser is just the same as using any Java servlet via a web server.

On invocation the web server launches the AprixConnect servlet and passes it the original request. AprixConnect then parses the request and returns an appropriate response.

There are currently three interfaces to AprixConnect: via a URL, POST inputs, or an embedded HTML tag <CONNECT>. XML will be included in a later version. All interfaces utilize tag attributes to define functionality as described below. The differences between the interfaces lie in how the interface is delivered to AprixConnect from the client.

## Interfaces

Connect elements are included within HTML documents to specify the location and layout of content to be merged within the document. When an HTML document is retrieved it is automatically parsed and all CONNECT tags are processed. The retrieved content is then merged into the document and returned to the client as a single document.

The Connect elements follow the syntax of HTML 4 tags as below:

```
<CONNECT {attributes}>{Failure text}</CONNECT>
```

If successful the body of the content identified by the tag replaces the tag. For HTML content this is the content of the BODY element of the page and does not include the HEAD element nor proceeding SCRIPT elements. Only the content of the BODY element is included – the BODY start and end tags are not merged.

If unsuccessful, the text contained between the start tag and the end tag is instead merged. This allows a designer to include additional text to be sent to the client should the operation fail. This text is optional and may contain other Connect elements. If the first Connect element fails AprixConnect will attempt to connect to the source indicated by the second embedded element. Connect elements may be nested in this way to any level.

AprixConnect will attempt to connect to the first specified database (say a transaction server). If the connection fails it will attempt to connect to a specified alternate database (a backup). If still unsuccessful it will merge a page from the Aprix site explaining that the server is down. Finally, if all attempts fail the text *"Could not connect to transaction server"* will be embedded instead.

Both the start tag and the end tag are compulsory. If not present the tag will not be processed and the tag is likely to be ignored by the browser.

Further discussion of the merging of content can be found in 'Merging Content'.

Commands to AprixConnect can be passed via the query string of the URL or POST through any browser. The attributes in the query string are interpreted and actioned in the same manner as the Connect element.

### For example:

<http://server/servlet/connect?FUNC=HTML&ACTION=http://www.myserver.com>

Retrieves the URL <http://www.myserver.com>

# Caching & Performance

## Overview

To improve performance the application will maintain an application cache. All retrieved content will potentially be stored in the cache for fast retrieval by later requests reducing the need to make connections to data sources where content has already been retrieved and is still fresh.

There are different types of data to be cached so ApatrixConnect will require two caches; an application wide site cache and individual session/user caches.

### Site Cache

The site cache will be a single generic cache for storing content from any backend system. Anything from HTML pages and images to complete recordsets or slices of customized application data could be stored in the cache for faster retrieval of shared content.

As the cache is application-wide and accessible from all sessions it may grow to be very large. Due to the variety of cache configurations and the importance of good performance a method must be provided to allow the user to tune the cache to a degree. For example, initial and maximum cache sizes (both memory and disk).

### User Caches

Sometimes content is tailored towards individuals users. For example, the front door of a site could be customised through Apatrix. The URL is the same for all users but the content returned is different. In this case the site cache could only store a "normal" uncustomised page. The customized pages would need to be stored in a user cache – a separate cache for each user (or session).

The user cache will be much smaller, typically only a few dozen pages.

## Cache Strategies & Control

In order to maintain a balance between excellent performance and fresh content a number of management strategies will be employed. This will ensure the cache will be flexible enough to provide the required balance whilst allowing for run-time fine tuning and further customizations in applications developed with this tool-kit.

These strategies revolve around two central questions: when and what to cache; and when to refresh a cached object.

The mechanisms for controlling caching are through storage and expiry attributes.

These attributes are defined in the following sections. They can be used for any type of content retrieval through either the URL, POST or CONNECT interfaces.

### When to Cache: The CACHE Flag

Whether or not to cache an object can be controlled via a flag in either the Connect tag or the query string depending on the interface used. Different values determine whether the object is to be cached and which cache to store it in. Objects not stored in the cache will always be retrieved from the original source.

### For example:

```
<CONNECT ACTION=http://www.apatrix.com CACHE=SITE>
```

In this example, the page <http://www.apatrix.com> will be retrieved and stored in the cache. Future requests will then respond with the cached page unless invalidated by the cache strategies outlined in the next section.

---

## When Cache Objects Are Refreshed: The EXPIRES Attribute

An obvious problem with caching lies with the question of the freshness of a page. How do we know that the page stored in cache is the same as the live page on the remote server? Can we be confident that the live page has not been updated since we last refreshed the cache's copy?

The EXPIRES attribute allows the developer to specify how long to maintain a document in cache before being removed. Once removed from the cache the next request for the document will force the document to be retrieved from the original server. This reduces the probability of delivering outdated content to a user.

### *Specifying an Absolute Time*

Values for the EXPIRES attribute can either specify a relative time or an absolute date. An absolute date specifies the date and time the document expires, for example "Mon, 29 May 2000 03:04:18 GMT". A request for this document after this time will delete the document from the cache and a new copy will be fetched.

The date specified should follow the general date/time format for your region. AptribConnect will use the regional date format settings of the operating system to parse an absolute date.

### **For example:**

EXPIRES="01/02/2000 3:47 AM"

Using the Australian date settings this would be interpreted as 1 Feb 2000. Under American conventions this becomes 2 Jan 2000.

Dates using a time zone (eg, "GMT") or specifying months using strings (such as "May") are also acceptable. Unless specified the default time zone of the operating system is used.

### *Specifying A Relative Time*

Rather than specifying an absolute time a relative time can be used to specify the document will expire some time after the document is placed in the cache, for example a number of hours or days. The actual time the document expires is then calculated from the time the document is retrieved and added to the cache.

By design only seconds, hours, days or months may be specified. Minutes are not supported to simplify the interface (M is used for months). Instead a multiple of seconds can be used (eg. 300 seconds for 5 minutes).

HTTP guidelines recommend documents are not given an expiry time greater than one year in the future. This is to promote the use of fresh content on the Web. As such the Year unit is not supported<sup>1</sup>.

## Cache Control through HTTP Headers

A lengthy discussion on the various caching strategies recommended for browsers, proxies and servers, including explanations of the cache control mechanisms employed within HTML and HTTP can be found in the [RFC 2616: HTTP/1.1](#) specification.

If there is a conflict between HTTP cache-control headers supplied by the remote server and AptribConnect cache attributes explicitly defined, AptribConnect attributes will take precedence. This is to allow the designer of an AptribConnect system the maximum configuration control. Some servers (like Domino) always specify no-cache thus forcing proxies to always retrieve new content. Overriding gives us the ability to cache pages we know won't change frequently thus improving performance on these servers.

If AptribConnect attributes are not specified via the interface then cache control will default to the HTTP standard.

---

<sup>1</sup> Whilst the Year unit is not supported AptribConnect does not force a maximum of twelve months for an expiry time. For example, an absolute date of 1 May 2040 or a relative date of 360 months are both acceptable to AptribConnect and will be used. The use of such expiry times by this method, however, is discouraged.

# Merging Content

## AprixConnect: A Smart Proxy

AprixConnect is, in essence, a highly configurable web server that retrieves input from different back-end servers in a variety of formats and returns back to the user via the HTTP 1.1 standard.

- Serve plain web pages from any web server.
- Reuse legacy components eg COM+/COM, EJB, JB
- Fetch data from multiple web servers and return a single HTML page.
- Send content via the HTTP POST method.

Return content from databases and other systems and return in a single HTML page.

## Retrieving Content

The method for retrieving content from a web server is as follows:

1. On invocation parse the interface to determine the required function and action (ie. A URL).
2. Check cache for content. If valid content found skip to step 4.
3. Connect to the server and retrieve the content.
4. Modify relative links within the document (See [0 Maintaining Links In Returned Content](#)).
5. Process CONNECT tags. Repeat steps 2 & 3 for each Connect tag.
6. Output merged content to the client.

## Maintaining Links In Returned Content

Links and references to other URLs within the content served by AprixConnect must be returned relative to the product. In this manner, when clicked the client application will still call our proxy to service the URL, rather than going straight to the URL native.

### For example:

Assume a user is accessing <http://www.myserver.com> via AprixConnect. The full URL to AprixConnect might be:

<http://localhost:8083/servlet/connect?FUNC=HTML&ACTION=http://www.myserver.com>

Suppose the URL <http://www.myserver.com> contains a relative link to the page *news.html*. In order for the link to *news.html* to be accessed via AprixConnect the URL of the link will need to resolve to

<http://localhost:8083/servlet/Delivery?FUNC=HTML&ACTION=http://www.myserver.com/news.html>

In this manner the client calls AprixConnect to process the request rather than requesting the page directly from [www.myserver.com](http://www.myserver.com) or failing with a *404 File Not Found* error.

Possible areas within an HTML page where links can be specified include:

- Anchor tags: `<A HREF="news.html">`
- Body tags: `<BODY BACKGROUND="background.gif">`
- Images: `<IMG SRC="arrow.gif">`
- Forms
- Link tag: `<LINK TYPE="text/css" HREF="mystyle.css">` (for style sheets)
- Frames: `<FRAME SRC="main.html">`
- Base tag: `<BASE HREF="http://yourserver/">`