

en|chrome

White Paper

© Arrak Software, March 2012

Oy Arrak Software Ab
Tillinmäentie 1 C
02330 Espoo
FINLAND

tel +358 (0)9 855 0365

<http://www.arrak.fi/enchrome>
<mailto:enchrome@arrak.fi>

Table of Contents

1	INTRODUCTION	3
1.1	TYPICAL WEB APPLICATIONS	3
1.2	ENCHROME	3
1.3	BENEFITS	3
1.4	REQUIREMENTS	3
2	APPLICATION ARCHITECTURE	4
2.1	THE BIG PICTURE	4
2.2	ENCHROME BUILDING BLOCKS	4
2.2.1	<i>Enchrome Expression Language</i>	5
2.2.2	<i>Ztags</i>	5
2.2.3	<i>Component Libraries</i>	5
2.2.4	<i>ZUL Files</i>	5
2.3	EXTERNAL CONNECTIVITY	6
2.4	SECURE COMMUNICATION	6
3	ENCHROME FEATURES	7
3.1	USER RIGHTS	7
3.2	SECURITY	7
3.3	PERFORMANCE	7
3.4	SIMPLE AND SECURE APPLICATION DEPLOYMENT	7
3.5	TRANSLATIONS	7
3.6	AUTOIMAGES	7
3.7	PROFILING	8
3.8	ERROR HANDLING	8
3.9	OTHER FEATURES...	8
3.10	CONCLUSION	8

1 Introduction

1.1 Typical Web Applications

Typical web applications consist of code that fetches data from databases and business logic components, formats and presents it to the user, and allows the user to edit and update data. Combining data extraction logic, data formatting and presentation, with all the other necessary things, such as user authentication, access checks, and HTML code, often results in “unmanageable spaghetti code”. Having the high-level application aspects such as program logic buried in lots of implementation details, increases both risk of errors and cost of maintenance and makes it extremely hard to reuse code.

1.2 Enchrome

Enchrome is a solution for rapidly creating advanced user interfaces in Internet-enabled applications by effectively separating program logic from the low-level implementation details.

Enchrome has been developed and used by Arrak Software and our partners since the late 1990s, so it is a mature technology that has been used in several dozen projects for tens of different customers in real production use. It is continuously being updated with new features and security enhancements.

The core functionality of enchrome allows UI components to exchange information with existing business logic and data sources. This information is presented to the end user in a client-specific format, usually HTML.

The developer is provided with a rich set of ready-to-use UI components, such as tabbed dialogs, search lists, popup-dialogs and different kinds of edit controls. In addition to these, the developer can create custom components to satisfy specific needs.

1.3 Benefits

Because of the high abstraction level of the UI components, HTML implementation details and graphic design aspects are separated from program design. This combined with component reusability and a development environment that needs no recompilation to see the results of a modification, means that applications can be developed very rapidly and easily maintained.

Applications using enchrome are deployed through a tamper-proof packaging mechanism, thus ensuring the integrity of the delivered software. An application built with enchrome can be shipped as a single application file, which is both encrypted and cryptographically signed.

Long-term benefits include short development cycles and high code reusability across different applications.

1.4 Requirements

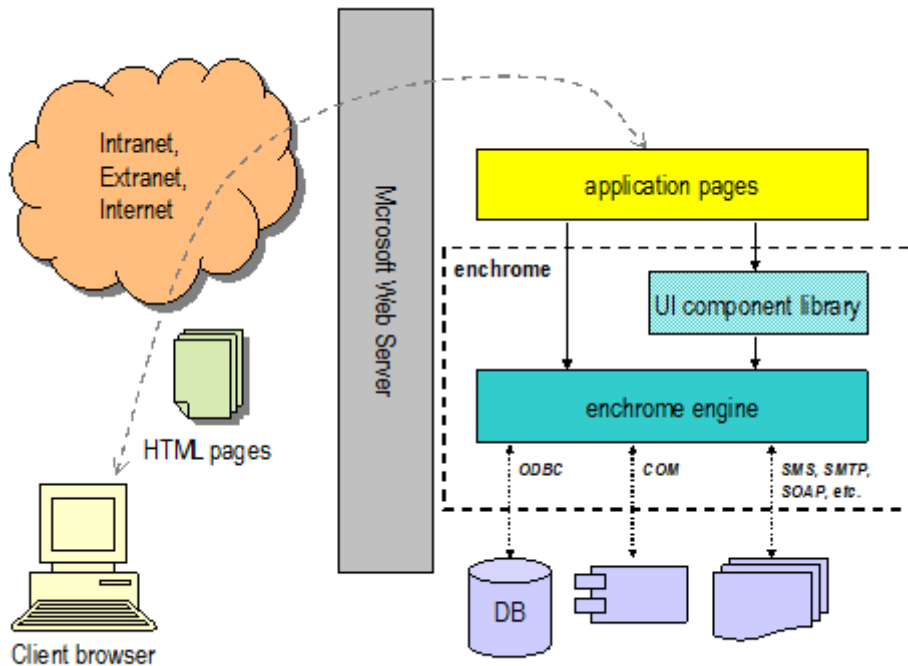
The Enchrome server runs under the Microsoft IIS Web Server (Windows 2000 or newer) as a native ISAPI DLL with no other modules needed. Therefore the IIS server can be stripped of all potentially insecure modules such as ASP, SSI, filters, etc.

Enchrome has been rigorously tested and used in production environments against Microsoft SQL Server and Sybase Adaptive Server Anywhere databases, but any data source with an ODBC driver can be used (e.g. Oracle, Access databases, Solid or even text/CSV files).

Clients using enchrome-based applications have absolutely no special requirements other than a web browser (or actually anything based on the HTTP/HTTPS protocol).

2 Application Architecture

2.1 The Big Picture



The enchrome engine runs as an ISAPI extension under Microsoft's web server (IIS), i.e. it consists of a single DLL file that is loaded by the IIS. The engine is responsible for the execution of the pages requested by the client, as well as handling of data connectivity, user authentication, access rights, sessions, caching etc. The request results in a page being generated, and depending on the UI component library used, can be e.g. an HTML page viewable in a standard web browser without the need for any special software.

The enchrome engine fully utilizes the multi-threaded architecture of the IIS, and is capable of very fast response times and high throughput by the use of powerful caching and precompiling of server pages.

2.2 Enchrome Building Blocks

When developing applications using enchrome, there are two different abstraction levels that can be used. The lower level is very similar to, e.g. ColdFusion, PHP or ASP, where an otherwise ordinary HTML page is extended with server-side directives that are executed within the HTML page. In enchrome, these pages are called *ZML pages*, and they can contain **ztags** and embedded **expressions**.

Programming on a higher level of abstraction is accomplished through the special *User Interface Description Language*, where user interfaces are described in a very brief XML notation with emphasis on structure, program logic and data connectivity. The user interface descriptions utilize **component libraries** for performing the actual work, which hides the technical implementation details and improves productivity, reusability and maintenance. User interface description files are called *ZUL files* and can contain one or several interface descriptions, with the possibility to reuse and extend interfaces in an object-oriented fashion.

The source pages can be edited with any text editor (or HTML or XML editor of your choice) and instantly be tested in a web browser, making it a true edit-and-run environment for rapid development and testing.

2.2.1 Enchrome Expression Language

The enchrome expression language is a scripting language that has been specially designed for making data manipulation and formatting both easy and intuitive to use. Its syntax is very similar to C and JavaScript, and contains almost 500 functions for doing a vast range of things from parsing and formatting to using databases and strong encryption.

The expression language has native support for using any scriptable (automated) COM object (also called ActiveX components), allowing third-party components to be easily used. This is also a very powerful mechanism for extending the expression language.

It is also possible to include script code written in VBScript or JScript.

2.2.2 Ztags

Ztags are special HTML-like tags that all have names beginning with the letter **z**. These tags are recognized by the enchrome engine, and the functionality they provide range from flow control to data connectivity.

In addition to the over 20 ztags already recognized by the enchrome engine, user-defined tags (procedures) can be defined and used in any ZML file, making procedural programming, abstraction and reuse as natural as in any programming language.

Example: A simple ZML page that fetches data from a database with an SQL query using the **zsql** tag and displays the result with a simple HTML table. The body of the **zsql** tag is repeated for each record in the result set. Embedded expressions start with a \$ sign, and the expression functions **curdate**, **iter** and **toupper** used in the example should be rather obvious!

```
<html><head><title>Example Page</title></head>
<body>
<h1>Product Listing at $curdate()</h1>
<table>
  <tr><th>ID#</th><th>Name</th><th>Quantity</th></tr>
  <zsql query="SELECT id, name, qty FROM products">
    ${bg = (iter()%2)==0 ? 'white' : 'cyan'}
    <tr bgcolor="$bg">
      <td>${toupper(id)}</td>
      <td>$name</td>
      <td align="right">$qty</td>
    </tr>
  </zsql>
</table>
</body></html>
```

2.2.3 Component Libraries

Component libraries are implemented with standard ZML files. Currently enchrome includes the *WebApp UI SDK* component library, for rapidly creating advanced user interfaces with a consistent look. The WebApp library contains a rich set of ready-to-use components for building tabbed dialogs, search lists, popup-dialogs, etc. Developers can create custom components for simple reuse of acquired knowledge.

The use of component libraries effectively separates the implementation of e.g. the graphical look from the application logic. This means the application developer can focus on the implementation of the functionality, while someone else (the component library developer) can be the expert on how to make good-looking HTML pages.

2.2.4 ZUL Files

In a ZUL file, library components are used with simple parameters to define the structure and logic of the user interfaces. Implementing a dialog for editing a selected record from the database, typically involves formulating the SQL query for fetching the record, and using selected field types to edit the specific fields.

Example: The following example is a fragment from a ZUL file (because it can define many related or unrelated user interfaces) that fetches the database record selected with a query argument, and displays a dialog for editing the fields of the record.

```
<dialog id="person"
  query="SELECT * FROM person
        WHERE Person_ID = ${ (int)input::key}">
  <cont_tab id="person" label="Person">
    <ctrl_edit id="First_name" label="First name"/>
    <ctrl_edit id="Last_name" label="Last name"/>
    <ctrl_separator/>
    <ctrl_edit id="Company" label="Company"/>
    <ctrl_edit id="Position" label="Position"/>
    <ctrl_separator/>
    <ctrl_address id="Address_ID"
      basename="address_record"/>
    <ctrl_separator/>
    <ctrl_submit id="submit" keyname="Person_ID"/>
  </cont_tab>
  <cont_tab id="optional" label="Optional Information">
    ...
  </cont_tab>
</dialog>
```

Referencing this dialog with an URL that looks something like
.../person?key=42 results in the following dialog:

Person		Optional information	
First name	<input type="text" value="Happy"/>		
Last name	<input type="text" value="User"/>		

Company	<input type="text" value="Oy Arrak Software Ab"/>		
Position	<input type="text" value="Testuser"/>		

Address	<input type="text" value="Ruukintie 20 C"/>	Phone	<input type="text" value="(09) 260 65 380"/>
	<input type="text"/>		<input type="text"/>
<input type="text" value="02330"/>	<input type="text" value="ESPOO"/>	Email	<input type="text" value="happy.user@arrak.fi"/>
<input type="text" value="FI - Suomi"/>			

<input type="button" value="Save"/>		<input type="button" value="Delete"/> <input type="button" value="Back"/>	

2.3 External Connectivity

In addition to the support for COM objects mentioned in section 2.2.1, enchrome can connect to any database with an ODBC interface. There is also built in support for the Sonera SMS gateway, sending of emails (with optional attachments) through any SMTP server, Microsoft's XML parser (MSXML 3.0 or higher) for efficiently running XSL stylesheet transformations, etc.

2.4 Secure Communication

Applications requiring secure client traffic, can simply install a standard SSL Server Certificate in the web server, making the all traffic to and from the enchrome-based application encrypted using the standard HTTPS protocol that works with any standard browser.

3 Enchrome Features

3.1 User Rights

Enchrome has native support for requiring user authentication with usernames and passwords. Users (and groups) can be assigned to groups, and different access rights to different logical parts of an application can be granted to both users and groups independently. By creating groups according to different roles used in an application and assigning role-based rights to the different parts of an application, implementing and managing rights for users is very easy.

Because authentication and access rights are a built-in feature of enchrome, no extra lines of code is needed in the application which keeps the program clear but also secure because the developer cannot forget to check the access rights.

3.2 Security

Designing secure web applications is one of the most important aspects of enchrome. Protection against injection attacks is easily done with the help of simple features in the expression language, high-level database functions guard against incorrectly formatted data, brute-force guessing of passwords is effectively stopped by built-in increasing delays, only well-formed requests are allowed (URLs may only contain certain legal characters), etc.

3.3 Performance

Enchrome is designed with performance as an important goal. This is accomplished through advanced caching of precompiled pages in the enchrome engine, database connection pooling and caching of frequently used data.

In addition to this, normal HTTP client-side caching is also automatically supported, lowering bandwidth use, server load and response times. Once a user has downloaded a page into his browser, his local copy of the page is reused until e.g. an update in the database causes the page to no longer be valid.

3.4 Simple and Secure Application Deployment

All the source pages for an application can be packaged into a single cryptographically secure file. This makes application installation extremely easy, because you only need the enchrome engine DLL-file and the application package (and configuration of database sources).

Your intellectual work is automatically protected, because your source code is not accessible, nor can the application be tampered with. It is, of course, also possible to run an application directly from the clear text source files, which is how you normally develop the application.

3.5 Translations

Many web applications need to be able to speak to the user in his own language. There is no need to keep multiple copies of the source files for different languages. Instead, the single set of source files can dynamically be translated into different languages, again keeping the program structure and logic separate from the implementation details.

Selected texts in the source files can have different language alternatives stored separately from the source file. The application developer never has to see or deal with the different translated texts, as he works on the single source file with the texts written in a language he understands.

All error messages built into enchrome can also be translated into any desired language.

3.6 Autoimages

The enchrome engine is capable of generating graphical buttons on the fly, with the text, font, size, colors, background template, etc. directed by parameters. A consistent graphical look of the

application is easy to obtain, even with dynamic texts. The texts on the graphical buttons can even be translated into the user's language.

3.7 Profiling

The en|chrome engine automatically gathers run-time profiling data internally. This data includes execution counts as well as minimum, maximum and average execution times. It is also possible to enable detailed execution tracing for requests, giving great insight into where and why time is spent during the processing of a request. All this profiling data helps the developer focus on the right parts of an application when aiming at further improving the performance of an application.

3.8 Error Handling

Customized error handling can be implemented by registering user-defined error handlers (implemented in ZML) for specific errors and situations where the built in error handling of simply aborting the execution with an error message is inadequate. The user-defined error handlers can ignore, change, log or report the error in any way desired, and further execution can be aborted or continued as appropriate.

3.9 Other Features...

Of course there is much more, such as logfiles, transactions, monitor sections, file inclusions, session-specific and global variable storage, file uploads, file system manipulation, unicode functions, cookie support, comments, output capture, system runtime information, etc.

3.10 Conclusion

All this is available right now, for a very developer-friendly licensing. Go to <http://www.arrak.fi/en|chrome> for more information.