

MARCO VON BALLMOOS

120-12 85th Avenue, Apt. 2E
Kew Gardens, NY 11415
718.441.6949 (Home)
212.529.3913 x150 (Office)
marco@earthli.com

Skills

Experience/Expertise Matrix

This matrix shows a detailed list of programming languages, libraries and APIs that I have used.

A level of 1-3 signifies beginner, 4-7 intermediate and 8-10 expert. Experience is measured in years.

<u>Programming</u>	<u>Level</u>	<u>Experience</u>	<u>Database</u>	<u>Level</u>	<u>Experience</u>
OO Design	9	8	MySQL/SQL	7	2.5
C++	9	6	SQL Server/T-SQL	7	1.5
MFC	8	6			
STL	7	3	Web		
Object Pascal	7	4	PHP	9	2.5
VCL	6	3	ASP	8	1.5
Win32	9	7	JavaScript	8	6.5
PowerPlant	6	2	CSS	9	5
MacOS Toolbox	7	2	HTML	10	6.5
Eiffel	6	5	DHTML	7	2
			Graphic Design	6	8
			Apache/MySQL Admin	6	2.5

Systems

Windows 3.x - 2000; MacOS 8.x - OS X

Tools

Metrowerks CodeWarrior for C++ 4.0 - 7.0; Borland Delphi 1.0 - 6.0; ISE Eiffel 4.5 - 5.1; Microsoft Visual C++ 4.0 - 6.0; Visual Parse++ 2.1 - 4.0; DreamWeaver 2.0 - 4.01; Photoshop 4.0 - 6.01; Perforce 99.1 - 2001.1

Work Experience

LOGICAT INC., NEW YORK, NY AUG 1994 - AUG 2002

Logicat Course Administrator May 2002 - Aug 2002

Designed and developed extensions to the CE Manager product to allow firms to track a user's CLE from start to finish, especially in the role of a course provider. Existing CE Manager features and new features were grouped into modules to allow deployment of different sets of software for different firms. All pages presented a dynamic view built from the features purchased by the firm combined with the rights of the logged-in user. Commands, menu items and lists of information available were all controlled by this feature/permission matrix. New functionality included:

- Scheduled courses with a calendar view and rich view of scheduled courses for each user
- Printable certificates for providers and individual users
- Course evaluation and results aggregation
- Authoring module for creating course evaluations

Logicat CE Manager Apr 2001 - Apr 2002

Designed and developed a web product that tracks continuing education credits for attorneys in New York and California using IIS 5.0 and SQL Server 2000. CE Manager includes:

- Validation for all data entry (users, courses, etc.).
- A user search and reporting tool that lets administrators create customized reports.
- Reminder and statement services for all licenses.

Logicat Online Testing Feb 2001 - Apr 2001

Worked with another developer to create a testing and test-authoring solution for web sites using Microsoft IIS 5.0 and SQL Server 2000. This was used later to build online learning solutions for clients.

One-on-One with the SAT 2.0 Jan 1999 - Feb 2001

Led a team of three developers to port the Logicat Test Engine to the MacOS using Metrowerks CodeWarrior 5.0.

- Designed and developed a cross-platform library with a listener/broadcaster event structure.
- Ported the ELF library to a custom cross-platform graphics library.
- Implemented a cross-platform file-format with endian support.
- Created new features such as arbitrarily nestable framesets and script/event driven controls.
- Added support for dynamic/downloaded content.

Logicat Test Engine Nov 1997 - Dec 1998

Led a team of three developers to design and develop a system to create tests/tutorials for multiple users for network or CD deployment on Windows using Visual C++ 6.0.

- Test items and pages are created in an RTF-capable word processor with embedded meta-tags.
- Item text can contain WYSIWYG text, paragraph and table formatting.
- Items are compiled and assembled in a database using improved ELF technology from the CPA software.
- Items can be scripted together with an event-based script language.
- Tutorials can be enhanced using sounds, images and videos.
- Tutorials can also gather and display user input and answers and can print or e-mail results.

Logicat CPA Review Software Update Release May 1997 - Nov 1997

Led a team of three developers to design and develop a study plan and calendar to integrate the CPA software with a live course using Visual C++ 5.0. Used Delphi 3.0 and OLE/COM to show PowerPoint presentations from the course.

Performance Transfer Utility/Installer/Technical Manual Feb 1997 - Apr 1997

Designed and developed a utility to manage a hardware copy protection library using Delphi 2.0 and incorporated it into an installer using Wise 5.0. Created a 100-page manual using HTML and CSS for the Technical Support department.

Logicat CPA Review Software Aug 1996 - Jan 1997

Incorporated ELF into the *PassMaster* code base as the main presentation and data storage engine. Created customized components descended from the ELF object library.

ELF Feb 1996 - Aug 1996

Designed and developed ELF, an object-oriented interactive document system in Borland Delphi 2.0 and Microsoft Visual C++ 4.1. Created EML, an HTML-like language, an RTF parser and a compound document format to convert existing data from RTF to ELF.

PassMaster CPA Review Web Site Dec 1995 - Jan 1996

Developed a web site for *PassMaster*, Logicat's CPA test preparation product. Used JavaScript to present a 40-question demonstration of the test preparation software.

One-on-One with the SAT Counselor Utility Sep 1995 - Nov 1995

Created a management and reporting utility for One-on-One with the SAT for multi-user networked school distributions using Borland Object Pascal 7.0.

One-on-One with the SAT Aug 1994 - Sept 1995

Designed and developed a test preparation program for the SAT in conjunction with The College Board using Borland Object Pascal 7.0.

EARTHLI.COM, NEW YORK, NY OCT 1999 - DEC 2001

earthli WebCore 2.0 Apr 2002 - Aug 2002

Upgraded the *WebCore* library to use a more well thought out, entirely class-based hierarchy. A new GUI library was built, with generic support for forms, grids, buttons, menus and more. All the GUI objects have customized descendents that handle the basic webcore objects and integrate nicely with the query library. Pages now access all functionality through application and page objects rather than a mass of global variables. This eases customization, extension and documentation.

earthli Projects Mar 2002 - Jul 2002

Developed a project manager based on the *WebCore*. Includes changes to record work completed and jobs to indicate work to do. Generates formatted changelists and tracks revisions.

earthli News Jan 2002 - Feb 2002

Rewrote the *Forums* as a *WebCore* application called *earthli News*. Once again, a large amount of code reuse helped create a stable, full-featured news service quickly and easily. Includes a subscription service.

earthli Recipes Jan 2002 - Feb 2002

Created a second *WebCore* application to manage recipes. Most of the functionality for this application was incorporated unchanged from the *WebCore*, like the user/security model, comments and full-text searching.

earthli Photo Albums 2.0 Oct 2001 - Dec 2001

Rebuilt the *earthli Photo Albums* to include many of the ideas developed in the *Forums*. Most of the actual technology was reworked and incorporated into a general-purpose back-end called the *earthli WebCore*. The *WebCore* is a content management system written in PHP that can be extended to serve many purposes. The Photo Albums are one such extension and they inherit many of their new features directly from the *WebCore*, such as enhanced security, nested albums, content management, anonymous users and comments.

earthli Themes Jun 2001 - Sep 2001

Redesigned the web site using only one set of styles, converting all existing content to a uniform namespace. Implemented several themes on top of this system that completely change the look (though not the layout) of the site.

earthli Forums Oct 2000 - Dec 2000

Designed and developed a bulletin board system using PHP 4.x and MySQL 3.x. It includes a flexible security system that describes rights for anonymous users, registered users, groups and individuals. Other standard features include full-text searching, nested replies and user profile

pages. In 2001, the system was updated to integrate with the *earthli Themes*.

earthli Photo Albums 1.0 Oct 1999 - Jul 2000

Designed and developed an online photo album using PHP 3.x and MySQL 3.x. The system included multiple users, albums, pictures, journal entries and a calendar.

History

CROSS-PLATFORM CONTENT ENGINE

I was first introduced to object-oriented programming in August of 1994 when I started working at Logicat. I had been hired straight out of college and was in charge of designing and programming an SAT test preparation program that would later become known as *One-on-One with the SAT*. I wrote the program in *Borland Object Pascal 7.0* with the part-time aid of another programmer. The final product was only part of the project, which also included a parser to convert RTF documents to a proprietary format and several indexing tools for building the content database that the program used. Soon after, I also wrote a utility for multi-user installations that allowed an administrator to maintain the user database and print progress reports. The utility was much easier to write because I was able to re-use much of the code from *One-on-One* itself.

Early in 1996, with *One-on-One* shipped and most support issues addressed, I wanted to focus on building software that wasn't so project-specific. I began to write a new document renderer which would be much more flexible and would concentrate only on displaying content and handling events. I wrote the initial prototype in *Borland Delphi 1.0* and christened it *ELF*. The prototype supported arbitrarily nested tables, images and multiple text styles.

ELF would see use in the next incarnation of Logicat's CPA test preparation software. First, I converted the prototype to *Microsoft Visual C++ 4.0* to ease integration into the existing code base. I also developed a very HTML-like document description language called *EML*, complete with a parser and an import filter for RTF documents. The RTF importer allowed content designers to use all of the sophisticated editing capabilities of Microsoft Word to arrange text. *EML* tags describe the structure, layout and display properties of document data. Some tags break a document into separate objects, whereas others describe the display properties of those objects. This would form the core of the program.

The *CPA Software* also extended the *ELF* library with objects that provided customized event handling. Much of the *CPA Software's* program logic was still based on existing code from previous versions of the *CPA Software*, but the content creation, file format and data rendering/event handling were all based on *ELF*. This project was also where I began using preconditions and postconditions, which I simulated in C++ using assertions.

Towards the end of 1997, the *CPA Software* had undergone more changes, but the *ELF* library was still the only truly reusable portion. Seeing a need for a more flexible tool that could address many different testing and training needs, Logicat set out to build a *Test Engine*. It was to serve as a development environment for building tutorials and testing materials and would have to be more flexible to cover a broader range of needs.

The first step in making a more general *Test Engine* was to move the content-specific program logic out of the executable. To this end, I designed a simple scripting language and built a compiler and interpreter for it. The language supports conditionals, loops and user-defined routines and has an easily extensible library of built-in routines to access engine functionality. Most of the user-defined routines serve as event handlers called by other components of the *Test Engine*. There were also some routines for manipulating the database of items created by the content creation tools. Combined with support for sounds, images and movies, a content designer could put together sophisticated presentations that varied widely.

All of the code for the *Test Engine* makes extensive use of preconditions and postconditions to ensure program correctness. The two other developers on the project would build layers of more specific

objects atop the basic engine and the assertions answered many of their questions before they were even asked. Since the engine was composed of so many components, I implemented a subscriber/publisher system that connected them together using abstract interfaces and kept them as separate as possible.

In January of 1999, Logicat began work on *One-on-One with the SAT 2.0*. The *Test Engine* was about to take its next step as they wanted this version to work on both Windows and Macintosh. Starting in January of 1999, I spent several months reworking the Windows version of the *Test Engine* to function as a cross-platform version with Windows extensions. At this point, I was developing in *Microsoft Visual C++ 6.0*. There were innumerable dependencies on Windows or library-specific classes and functions that had to be changed or eliminated. The MFC CString class was replaced with a version based on the C++ Library string class. Direct calls to determine screen size or amount of memory available were replaced with DISPLAY and SYSTEM classes, respectively. At each juncture, I put as much functionality as possible into cross-platform code and designed my classes so that the platform-specific version was as thin as possible.

Once a rudimentary version of the *Test Engine* was functioning in this way, we moved the code to the Macintosh using *Metrowerks CodeWarrior 4.0/5.0*. With the *Test Engine* using only platform-independent interfaces for system components, it was simple to develop Macintosh versions of system objects and snap them into place. Very quickly, the Macintosh version caught up to the Windows version in functionality. The final product had most of the code in a cross-platform *Test Engine* that encapsulated all of the program logic, with platform-specific plugins filling in functionality. In the end, we had to write classes to handle much of a common GUI program's functionality: file handling (based on C++ Library), file system, 2-D graphics, sound, movies, printing, timers and window manager. The data format was made compatible across platforms.

To support the feature set of a large program like *One-on-One*, the *Test Engine* was expanded to support an arbitrary number of windows, frames within frames (like HTML) and much more dynamic content.

The final product is a very web browser-like product that runs on both Windows and Macintosh natively, with no performance advantage on either platform and about 90% code shared. Most importantly, the *Test Engine* has no code customized to *One-on-One Version 2.0*. All functionality and content of *One-on-One* is defined in RTF data files, a script file and supporting media like pictures, sounds and movies and is completely separated from the *Test Engine* code. Thanks to assertions, there was far less of a need for debugging than on previous projects and the completion of *One-on-One Version 2.0* was far easier than *One-on-One Version 1.0*.

WEB-BASED CONTENT MANAGEMENT

I started writing web applications in October 1999 on my own web site, earthli.com. After some research, I chose the now familiar Apache/PHP/MySQL development environment. My first real application was a photo gallery to share vacation photographs online. Each photo album belonged to a user, who was able to edit it when logged in. The public could browse through the entire album with read-only access. Later, a list of editors was added to albums to allow collaboration between users.

My next project was a project manager for keeping track of change lists, revisions and build history for various projects. It occurred to me that this project and the previous photo albums shared many similar characteristics, which would be required of almost any collaborative web application. Each required user accounts, user validation and rights management, support for user comments and so on. I began creating these shared components with the goal of building the project manager on top of them.

From this work grew the *earthli Forums*, which implemented a robust security model, nested folders, nested comments, anonymous user tracking etc., but still wasn't built generically enough; it didn't allow easy adaptation to other applications. Late in 2001, I began work on the *WebCore*, which would incorporate the code from the *Forums*, but adapted to a more generic framework and whose API would be defined by a class-hierarchy.

The *WebCore* has several properties:

- Content is treated as contained within a nested hierarchy of folders. Folders can inherit rights from their parents. Any application can define multiple types of objects.
- Anonymous browsing is supported seamlessly with logged-in users.
- The security model is enforced within the library, returning only data that the browser has been validated to see. Users can be granted individual rights to create, modify, delete, purge content or modify rights within folders.
- Folder level rights can be granted to anonymous user, registered users, to a group of users or to individual users.
- User level rights can be granted or denied regardless of folder to ban users or create administrators.
- Every piece of data in the system is exposed as an object. Each object records its creation and modification information. Objects are retrieved using query objects that handle pagination and rights enforcement.
- Proper separation of code and display logic is emphasized. The web pages themselves deal only with PHP objects; queries are made through an object layer, so the pages themselves are not only unaware that a MySQL database is used, they are unaware that an SQL database is being used at all. In many cases, other PHP objects, like grids and trees, handle display and HTML generation as well, .
- Full-text searching is supported in the query object.
- Several other classes are available, including a browser class for determining the user agent, a text formatter class for displaying HTML data and a grid class that works with the query classes to paginate objects into a grid.

Version 2.0 of the *Photo Albums* was built on top of the *WebCore* at the same time as the *WebCore* itself was being developed. Soon after, the second application to be built with the *WebCore* was *earthli Recipes*, which was ready for testing and demonstration in just over a week. During development of the *Recipes*, I created a set of pages that handled setting user and folder rights and creating, editing, deleting and viewing comments that could be used 'as is' if no customization was required by the application. This would help bring future application up to speed even more quickly.

earthli News followed soon after and allowed me to retire the old *Forums* software and use only applications build with the *WebCore*. The fact that the *WebCore* handled the bulk of the processing, validation and querying left me free to focus on designing a better interface for presenting news. Also, fixed bugs and new features are applied to all earthli applications simultaneously because of the large amount of shared code. My latest *WebCore* application is creating the project manager I started over a year ago. Once again, it's off to a quick start because so much of the functionality is already available in the *WebCore*.

With the *WebCore* stabilized at version 1.1, development on the next version began in mid-April and continued until mid-August of 2002. This version had several goals in mind:

1. Move more of the display code into a class-based GUI library that would handle forms, grids, menus and more.
2. Standardize the API, using similar feature names where possible and classes to encapsulate all common tasks.
3. Replace the labyrinthian startup code for the webcore with a simple class-based, easily customizable interface.
4. Remove dependencies on global variables, either constants, automatic PHP variables or PHP special handling (like magic quotes).
5. Allow a page to hold objects from multiple webcore applications at once (remove reliance on shared constants and give all objects a reference to the application environment in which they run).
6. Clean up the file hierarchy to facilitate learning and documentation.

The first release was version 1.2, to which all existing applications were upgraded in mid-July. This release incorporated the new gui library; the forms library was particularly useful for encapsulating a lot of shared code and fixing many validation errors that existed. The next release was version 2.0, which changed the entire framework to use two global objects, a PAGE object and an

APPLICATION object. All other objects, be they queries, gui objects, application objects, etc. are retrieved from these two objects using an intuitive API. The move to this simpler, more powerful framework allowed goal 4 above to be satisfied more easily and, since all code is now encapsulated, a page can reference multiple *WebCore* applications, satisfying goal 5 above.

These updates haven't changed the functionality of the *WebCore* applications on the surface, but the increased code-sharing and more maintainable code fixed many errors and bugs and makes it much easier to create new functionality going forward. Even more, the library is far more usable and extendable, since the earthli-specific code has been extracted and re-cast as a layer on top of the new library, moving the *WebCore* that much closer to being released as a product.

Education

Hamilton College, Clinton, New York

Bachelor of Arts in Mathematics in May, 1994
Major: Mathematics; Minor: Physics