# Aspects of the integration of two or more web-applications

When integrating a web-application into another, there are several aspects. This document describes these aspects, possible obstacles and solutions thereof. The findings are based on the process of making Gallery 2, a PHP-based photo album management software, ready for interoperability with other web-applications.

- Principles
- Master-Slave relationship and Communication
- User and Group management
- Session management
- Authentication
- URL generation
- Visual Integration
- (Permission Management)
- Controlling Access
- Search and other API interfaces
- Namespace Collisions
- Environment

## Principles

- No forking: All applications can use their won upgrade path. No (developer) resources are wasted with forks. Glue code and APIs are used to glue the applications together
- Seamless integration: Make the resulting solution look and act like a single application
- Data integrity: Keep the application in sync', at any time. No periodic tasks (cron, scheduler, ..) should be necessary
- Make the life of integration authors as simple as possible: The more code has to be written for a specific integration, the longer it takes to write it and the more code must be maintained.

## Master-Slave Relationship and Communication

- Primary application is the master
- “Embedded” application is the slave
- Alternatives: duplex communication
- More than two applications: multiple master-slave relationships
- Master-Slave relationship throughout all aspects
- Communication between the two applications: PHP and on the same web-server. Or XMLRPC as a more powerful alternative

## User and Group Management

- **Tight vs. loose integration**: tight = both applications share the same database user table, loose = both applications use their own user management, but they are loosely coupled and never out-of-sync’. Standardizing user management to for tight integrations seems to be impossible. We explored both directions and decided us for a loose integration.
- **Event-based synchronization**
- On-the-fly user generation as low-tech fallback solution
- Periodic synchronization as low-tech fallback solution (update, delete users)
- Initial user and group synchronization (import / export)
- Groups not that important, possible obstacles
- Single new user registration solution (event-based synchronization or low-tech fallback)
- Unhandled: Adding registration user data fields for the other application

## Session Management

- A standardized session interface / API would be good.
- For the lack of such a standardization and since not all applications allow adding session data etc., parallel session management was chosen (master-slave), which actually means that both applications use their own session management
- Fun with cookies (see document about cookie management in G2)
- Session Timeouts

## Authentication

- Master does the authentication, slave trusts the master
- No-overhead solution for loading the corresponding user in slave (no additional db queries, ... necessary)
- Clear-text / Hashed passwords only important when accessing both applications directly
- Single-Sign-On (SSO) solution

## URL Generation

- URLs of the embedded application
- What parameters are required
- Short URLs / mod_rewrite / ...

## Visual Integration

- Central template system is unrealistic
- Slave returns generated HTML, CSS, JS, title, <head> content separately
- Master must have the possibility to add external CSS / JS <head> content to the page output, also a method to set the page title
- Master must have the option to allow the slave to output directly to the browser for special cases, e.g. DownloadItem in Gallery 2
- Removing specific menu items, e.g. registration / login links, ...
- Alternative: Standardized templating or widget-based output

## Permission Management

- Not integrated. Permission systems are too different and a central GUI to administer all permissions seems to be a too small benefit. Research needed.
- Both applications manage permissions separately, but they are still linked since both applications assign the permissions to the same users / groups.
- The master application can restrict access to the embedded application, the embedded application manages the detailed privileges

## Controlling Access

- Should both applications still be accessible directly, or is one of them only allowed to be accessed embedded
- Are there exceptions (G2: GR, DownloadItem, ..)

## Exposing Search Functionality and Other API

- Syndicate the search function, API required

## Namespace and Code Collisions

- Use prefixes for your class names
- Use prefixes for your db tables / columns
- Use a minimum of global variables and make sure their name is prefixed or very unique
- Use prefixes in constants
- When using 3rd party libraries like adodb, smarty, ... make sure to use the latest versions and make sure they work when included multiple times.
- Also make sure to use absolute file system paths in all includes. This is generally a good idea and even more important for interoperability

## Environment

- Character Encoding. Either all UTF-8 or the applications need to be flexible about input (from browser, other application) / output (HTML) character encoding
- PHP environment: register_globals, error reporting, PHP versions, ...