# Devel Documentation

SaltOS 4.0 r2067

May 2025

# Contents

# 1 Introduction

SaltOS4 is a modular, extensible framework for building Rich Internet Applications (RIAs). It is designed for rapid development of dynamic, offline-capable applications using a clean separation between backend and frontend.

It is composed of:

- A PHP backend (‘api/‘)
- A JavaScript frontend (‘web/‘)
- REST/JSON API with support for CLI execution
- Offline operation through a Service Worker proxy
- Declarative and dinamic app definitions via XML and/or YAML
- Shared templates and logic to reduce boilerplate
- Clean separation of frontend and backend
- Rapid development of apps with shared templates and logic

## 1.1 Advanced System Features

SaltOS4 includes two key functionalities that stand out for their utility and focus on traceability:

- Version traceability of records: Each modification in application data generates a new version of the record, maintaining a complete change history, similar to version control systems like Subversion or Git.
- Data access logging: The system records who accessed a piece of data, from where, and in what context (e.g., if it was shown in a list or form). This provides full auditability of data usage and consultation.

## 1.2 Project Structure

The root directory of SaltOS4 is organized as follows:

- ‘README.md‘: Provides an overview and quick-start instructions for the project.
- ‘LICENSE.md‘: Text of the open-source license under which SaltOS4 is released.
- ‘makefile‘: Automates repetitive development tasks, such as generating documentation using the scripts in ‘scripts/‘.
- ‘code/‘: Contains the full source code of the system, split into backend (PHP) and frontend (JavaScript) modules.
- ‘docs/‘: Documentation files written in ‘.t2t‘ format, including generated outputs (‘.pdf‘, ‘.html‘) and the source files used to build them.
- ‘scripts/‘: Collection of PHP utilities that extract documentation from code comments, produce ‘.t2t‘, ‘.pdf‘, or ‘.html‘ outputs and more...
- ‘ujest/‘: JavaScript unit tests using Jest.
- ‘utest/‘: PHP unit tests using PHPUnit.

This structure keeps the system cleanly separated between source code, documentation, automation, and testing components.

## 1.3 Code Directory

The ‘code/‘ directory contains the source code and runtime data of SaltOS4, structured as follows:

- ‘api/‘: Backend core written in PHP. Handles REST endpoints, internal services, and logic.
- ‘apps/‘: Application modules. Each app includes its own backend, frontend, and configuration (PHP, JS, XML).
- ‘data/‘: Stores persistent and temporary data generated by the system, such as files, images, downloaded or sent emails, cache, and temp files.
- ‘web/‘: Frontend resources — primarily JavaScript and CSS — used in the browser interface.

This structure supports both application logic and data flow across the SaltOS4 system.

## 1.4 Data Directory

SaltOS4 stores runtime and persistent data under the ‘data/‘ directory. This folder contains several subdirectories used for caching, file storage, logging, and background task management. It is essential for the proper operation of the system.

- ‘cache/‘: Stores cached data to improve performance.
- ‘cron/‘: Contains execution records and temporary files for scheduled tasks.
- ‘files/‘: Persistent file storage used by applications.
- ‘inbox/‘: Incoming data such as fetched emails.
- ‘logs/‘: Application and system logs.
- ‘outbox/‘: Outgoing data, such as email messages.
- ‘temp/‘: Temporary files used during various operations.
- ‘trash/‘: Items marked as deleted but not permanently removed.
- ‘upload/‘: Staging area for newly uploaded files before they are processed.

Make sure this directory is writable by the web server and CLI user.

# 2 Backend (api/)

The backend is under the ‘api/‘ folder and contains four folders (autoload, database, lib and actions) to organize the code depending their usage and functionality.

## 2.1 Autoloaded Modules (‘php/autoload/‘)

Core functions automatically loaded on each request:

- ‘autoload/apps.php‘: * Apps helper module
- ‘autoload/array.php‘: * Array helper module
- ‘autoload/compat.php‘: * Compatibility helper module
- ‘autoload/config.php‘: * Config helper module

- ‘autoload/database.php‘: * Database helper module
- ‘autoload/datetime.php‘: * Datetime helper module
- ‘autoload/error.php‘: * Error helper module
- ‘autoload/exec.php‘: * Execution helper module
- ‘autoload/file.php‘: * File utils helper module
- ‘autoload/getdata.php‘: * Get data helper module
- ‘autoload/gettext.php‘: * Gettext helper module
- ‘autoload/iniset.php‘: * Iniset helper module
- ‘autoload/json.php‘: * Json helper module
- ‘autoload/log.php‘: * Log helper module
- ‘autoload/memory.php‘: * Memory helper module
- ‘autoload/mime.php‘: * Mime helper module
- ‘autoload/output.php‘: * Output helper module
- ‘autoload/pcov.php‘: * PCOV helper module
- ‘autoload/perms.php‘: * Permissions helper module
- ‘autoload/random.php‘: * Random helper module
- ‘autoload/semaphores.php‘: * Semaphore helper module
- ‘autoload/server.php‘: * Server helper module
- ‘autoload/sql.php‘: * SQL utils helper module
- ‘autoload/strings.php‘: * String utils helper module
- ‘autoload/system.php‘: * System helper module
- ‘autoload/tokens.php‘: * Tokens helper module
- ‘autoload/user.php‘: * User helper module
- ‘autoload/version.php‘: * Version helper module
- ‘autoload/xml2array.php‘: * XML to Array helper module
- ‘autoload/yaml.php‘: * Yaml helper module
- ‘autoload/zindex.php‘: * Main execution module

## 2.2 Database Drivers (‘php/database/‘)

Supports:

- ‘database/libsqlite.php‘: * SQLite3 functions library
- ‘database/mysqli.php‘: * MySQL improved driver
- ‘database/pdo_mssql.php‘: * PDO MsSQL driver
- ‘database/pdo_mysql.php‘: * PDO MySQL driver
- ‘database/pdo_sqlite.php‘: * PDO SQLite driver

- ‘database/sqlite3.php‘: * SQLite3 driver

## 2.3 Libraries (‘php/lib/‘)

Not autoloaded; provide extra functionality:

- ‘lib/actions.php‘: * Actions module
- ‘lib/array2xml.php‘: * Array to XML helper module
- ‘lib/ascii.php‘: * Make Table ASCII
- ‘lib/auth.php‘: * Login functions
- ‘lib/barcode.php‘: * Barcode helper module
- ‘lib/browser.php‘: * Browser helper module
- ‘lib/captcha.php‘: * Captcha helper module
- ‘lib/color.php‘: * Color helper module
- ‘lib/control.php‘: * Control helper module
- ‘lib/cron.php‘: * Cron utils helper module
- ‘lib/dbschema.php‘: * Database schema helper module
- ‘lib/depend.php‘: * Dependencies feature
- ‘lib/export.php‘: * Export helper module
- ‘lib/files.php‘: * Files module
- ‘lib/gc.php‘: * Garbage collector helper module
- ‘lib/gdlib.php‘: * GD utils helper module
- ‘lib/geoip.php‘: * GeoIP helper module
- ‘lib/help.php‘: * Help feature
- ‘lib/import.php‘: * Import file helper module
- ‘lib/indexing.php‘: * Make index helper module
- ‘lib/log.php‘: * Log helper module
- ‘lib/math.php‘: * Math utils helper module
- ‘lib/notes.php‘: * Notes module
- ‘lib/password.php‘: * Password helper module
- ‘lib/pdf.php‘: * PDF helper module
- ‘lib/push.php‘: * Push utils helper module
- ‘lib/qrcode.php‘: * QRCode helper module
- ‘lib/score.php‘: * Score image helper module
- ‘lib/security.php‘: * Security helper module
- ‘lib/setup.php‘: * Setup helper module
- ‘lib/trash.php‘: * Send file to trash

- ‘lib/unoconv.php‘: * Unoconv library
- ‘lib/upload.php‘: * Add upload file
- ‘lib/version.php‘: * Version helper module

## 2.4 Action Modules (‘php/action/‘)

Handle concrete system actions (CLI-aware where needed):

- ‘action/add.php‘: * Add log action
- ‘action/app.php‘: * Application action
- ‘action/auth.php‘: * Authentication helper module
- ‘action/cron.php‘: * Garbage Collector action
- ‘action/gc.php‘: * Garbage Collector action
- ‘action/image.php‘: * BarCode action
- ‘action/indexing.php‘: * Make indexing action
- ‘action/integrity.php‘: * Make indexing action
- ‘action/push.php‘: * Garbage Collector action
- ‘action/setup.php‘: * DB Schema action
- ‘action/upload.php‘: * Add files action

## 2.5 Other API Components

This section lists complementary elements in the backend that support the main API logic.

- ‘index.php‘: Main API entry point. It loads required bootstrap files (‘autoload‘, ‘zindex.php‘) and dispatches incoming REST or CLI requests.
- ‘img/‘: Contains SaltOS4 logos and branding assets.
- ‘locale/‘: Stores multilingual resources used for translations and documentation. Includes ‘.yaml‘ files for language mapping, and ‘.odt‘/‘.pdf‘ for external formats.
- ‘xml/‘: Holds static configuration files (e.g. schema, themes, locales).
- ‘lib/‘: Contains third-party or integrated libraries used across the backend logic. Includes:
  - ‘atkinson‘, ‘gorrisans‘: font support
  - ‘browscap‘: browser capability detection
  - ‘edifact‘: EDI message parsing
  - ‘fpdi‘: PDF manipulation and import
  - ‘phpgeoip‘: IP geolocation
  - ‘phpspreadsheet‘: spreadsheet creation and parsing
  - ‘roundcube‘, ‘tcpdf‘, ‘wolfsoftware‘, ‘yaml‘: other protocol, rendering, and utility libraries

## 2.6 Configuration Files

SaltOS4 relies on several XML configuration files to define system-wide behavior, themes, languages, scheduled tasks, and database initialization. These files are stored in the root configuration directories and are essential for proper system setup.

### 2.6.1 dbschema.xml

Defines the database schema structure used across all applications. It lists all tables, their fields, types, primary keys, indexes, and relationships.

### 2.6.2 dbstatic.xml

Populates static content in the database. This includes predefined record values required for the system to operate correctly (e.g., permission types, core options).

### 2.6.3 config.xml

Contains global system configuration, such as paths, timezones, defaults, UI settings, and other bootstrap parameters.

### 2.6.4 cron.xml

Specifies scheduled tasks and their timing. Each entry defines a script to be executed periodically, along with interval definitions and optional conditions.

### 2.6.5 locale.xml

Defines available languages and translation mappings for the interface. Used by the frontend and backend to provide multi-language support.

### 2.6.6 bs_theme.xml

Defines Bootstrap theme settings, including color schemes, spacing, and variables that adapt the interface visually.

### 2.6.7 css_theme.xml

Defines custom CSS style variants used in the frontend. Enables additional visual personalization beyond Bootstrap presets.

## 2.7 API Access

SaltOS4 supports access via HTTP or CLI.

### 2.7.1 Web server access (Apache, Nginx, Cherokee)

The main idea of sending information to SaltOS is to use the latest technologies, to do it, we are using restful request

- GET with REST:
  - An example request: 'https://host/?/app/invoices/view/2'
  - '@rest/0' = app
  - '@rest/1' = invoices
  - '@rest/2' = view
  - '@rest/3' = 2
- POST with JSON:
  - An example request: 'https://host/?/app/invoices/insert'
  - Use 'Content-Type: application/json'
  - Body parsed into '@json/...'

### 2.7.2 Command-line access (CLI)

- 'php api/index.php app/customers/view/100'
- 'user=admin php api/index.php app/customers/view/100'
- 'cat data.json | user=admin php app/customers/insert'

## 2.8 API Authentication

SaltOS4 implements token-based authentication for both HTTP and CLI access. The authentication logic is handled by 'auth.php', which delegates to utility functions in 'php/lib/auth.php'. Tokens are validated based on IP address and user agent.

Supported authentication actions:

- 'auth/login': Authenticates a user using 'user' and 'pass' (JSON fields). Returns a token and user metadata.
- 'auth/check': Validates the current token.
- 'auth/logout': Invalidates the current token and ends the session.
- 'auth/update': Changes the password of the currently authenticated user.

Each request is handled in 'api/php/action/auth.php', which delegates to utility functions defined in 'php/lib/auth.php'.

The token is associated with the client IP address and user-agent to prevent misuse. Internally, 'current_token()' retrieves the token from the request and validates it against stored sessions.

### 2.8.1 HTTP Access

Users authenticate by calling the ‘auth/login‘ endpoint with a JSON payload and receive a token in response. This token must be included in subsequent requests using the ‘Authorization‘ header.

Example of authenticating via ‘curl‘:

```
curl -X POST https://yourdomain/api/auth/login \
    -H "Content-Type: application/json" \
    -d '{"user":"admin", "pass":"secret"}'
```

Once authenticated, subsequent requests must include the token using a header:

```
curl https://yourdomain/api/app/yourapp/list \
    -H "Authorization: Bearer your-token-goes-here"
```

The backend automatically links the token to the user and group using functions from ‘autoload/user.php‘. Permissions are enforced using ‘autoload/perms.php‘, which determines whether a user can perform a given action on a given application or record.

### 2.8.2 CLI Access

Authentication also works from the command line using the same API structure. Credentials are provided as JSON via stdin, and the result is a token.

Example::

```
echo '{"user":"admin","pass":"secret"}' | php index.php auth/login
```

Subsequent authenticated requests use the token as an environment variable:

```
token=your-token php index.php app/customers/list
```

Alternatively, if the user executing the PHP script is the owner of the ‘index.php‘ file, the token is not required. You can specify the user directly:

Example::

```
user=admin php index.php app/customers/list
```

This shortcut is only allowed for the instance owner. Other system users must use token-based authentication to prevent privilege escalation.

## 2.9 Token Lifecycle & Session Security

SaltOS4 uses token-based authentication for both HTTP and CLI access. Beyond basic login/logout operations, the system enforces session integrity and security through token validation mechanisms.

### 2.9.1 Token Binding

Each token is bound to the following client attributes:

- **IP address** (‘remote_addr‘)
- **User agent** (‘user_agent‘)

This ensures that a token cannot be reused from another location or browser, protecting against token theft and reuse attacks.

### 2.9.2 Expiration and Invalidation

Tokens are subject to expiration based on system configuration. They may be invalidated:

- After a certain period of inactivity
- On user logout
- When explicitly reset by administrators
- If the IP address or user agent changes

Expired or invalid tokens are automatically purged during background cleanup tasks (‘crontab_users()‘).

### 2.9.3 CLI Considerations

When executing scripts from the command line:

- Tokens can be obtained via ‘auth/login‘ and passed as ‘token=...‘ in subsequent commands.
- If the script is executed by the **owner of ‘index.php‘**, tokens can be bypassed by specifying ‘user=...‘ directly.
- This bypass is not allowed for other system users, providing a secure fallback that respects file ownership.

### 2.9.4 Additional Protections

- All token operations are guarded by semaphores to prevent race conditions.
- The backend checks token validity on every request via ‘checktoken()‘ (web) or ‘current_token()‘ (CLI).
- Sessions are tightly coupled to server-side validation and cannot be spoofed via headers alone.

These security measures collectively ensure that only valid users can perform actions and that token reuse or hijacking is effectively prevented.

# 3 Frontend (web/)

Frontend is a JavaScript SPA in the ‘web/‘ folder.

- ‘index.htm‘: loads Bootstrap, SaltOS scripts
- ‘web/js/‘: client-side modules
- ‘web/lib/‘: JS libraries

JavaScript Modules:

- ‘app.js‘: * Application helper module
- ‘auth.js‘: * Authentication helper module
- ‘backup.js‘: * Backup & Autosave helper module

- ‘bootstrap.js‘: * Bootstrap helper module
- ‘common.js‘: * Common helper module
- ‘core.js‘: * Core helper module
- ‘driver.js‘: * Driver module
- ‘filter.js‘: * Filter module
- ‘form.js‘: * Form helper module
- ‘gettext.js‘: * Gettext helper module
- ‘hash.js‘: * Hash helper module
- ‘object.js‘: * Object helper module
- ‘proxy.js‘: * Proxy module
- ‘push.js‘: * Push & favicon helper module
- ‘storage.js‘: * Token helper module
- ‘token.js‘: * Token helper module
- ‘window.js‘: * Window helper module

## 3.1 Logic (core.js and app.js)

SaltOS4’s frontend architecture is built upon two central modules: ‘core.js‘ and ‘app.js‘. Together, they form the foundation for most runtime behavior, enabling dynamic interaction with the backend and managing the user interface.

### 3.1.1 core.js Overview

The file ‘core.js‘ contains low-level utility functions and system-wide features. It is responsible for:

- Capturing and reporting JavaScript errors, including sourcemapped stack traces.
- Sending AJAX requests, abstracted through ‘saltos.core.ajax(...)‘.
- Creating and manipulating HTML and DOM nodes (‘saltos.core.html(...)‘, ‘saltos.core.append(...)‘).
- Managing configuration options and global parameters.
- Serving as a base for other frontend modules (like auth, filter, window, etc).

This file acts as the technical backbone for the client-side logic.

### 3.1.2 app.js Overview

The file ‘app.js‘ implements the high-level application layer. It builds on top of ‘core.js‘ and adds features related to UI interaction and view management:

- Registers and manages application views and layouts.
- Handles modal dialogs, toast notifications, and main layout rendering.
- Processes hash-based navigation (‘#app/xyz/...‘) and dispatches actions.

- Manages event hooks (e.g., ‘saltos.app.login‘, ‘saltos.app.ready‘).
- Integrates layout types (‘type1‘ to ‘type5‘) through ‘saltos.driver‘.

It is also responsible for launching the application when loaded in the browser.

### 3.1.3 Integration and Responsibilities

- ‘core.js‘ provides shared functionality, DOM helpers, AJAX, and error reporting.
- ‘app.js‘ drives the user interface, manages state, and coordinates navigation.
- Modules like ‘auth.js‘, ‘filter.js‘, ‘form.js‘ and others rely on both core and app for foundational logic.

These two modules form the base platform for all SaltOS4 frontend applications.

## 3.2 Deployment

To use SaltOS4 from the browser:

- Publish ‘web/‘
- Create symbolic links inside ‘web/‘:
  - ‘apps/‘ → application resources
  - ‘api/‘ → backend
- Inside ‘api/‘, symbolic links to:
  - ‘data/‘ → file storage
  - ‘apps/‘ → app definitions

## 3.3 Offline Support (Service Worker)

SaltOS4 includes a service worker defined in ‘js/proxy.js‘ that transparently enables offline support and network optimization. It plays a key role in improving performance and user experience, even under poor connectivity conditions.

### 3.3.1 Main Features

- Intercepts all ‘fetch‘ requests and serves them from cache when offline.
- Queues write operations (e.g., POST) and synchronizes them when the network is restored.
- Integrates seamlessly with the core AJAX layer (‘core.js‘), requiring no changes in application logic.
- Logs operations (network, cache, queue, error) using visual debug tools.
- Handles smart caching and fallback strategies depending on the request type.

### 3.3.2 Internal Structure

The service worker (‘js/proxy.js‘) contains the following key components:

- **‘console_log(message)‘**:
  - Broadcasts messages to all connected clients using ‘postMessage‘, since ‘console.log()‘ is often suppressed in service workers.
- **‘debug(action, url, type, duration, size)‘**:
  - Builds color-coded debug output showing the nature of the request (‘network‘, ‘cache‘, ‘queue‘, ‘error‘), its duration, and response size.
- **‘fetch‘ event listener**:
  - Intercepts all network requests.
  - Determines behavior based on request method and headers.
  - Responds from cache, fetches from the network, or queues operations depending on availability and context.
- **‘message‘ event listener**:
  - Handles external control commands sent by the application. Supported messages include:
    - ‘resetCache‘: clears the entire cache storage.
    - ‘resetQueue‘: clears the queue of pending offline operations.
    - ‘stop‘: unregisters the service worker.
    - ‘hello‘: returns a ”hello” reply for testing communication.
    - ‘sync‘: processes all queued requests and tries to send them online.

These capabilities allow external scripts to manage the state and behavior of the proxy dynamically.

### 3.3.3 Debugging and Observability

To aid debugging:

- All major proxy actions are logged with clear visual indicators.
- Logs are forwarded to client tabs for visibility, even if the browser suppresses SW console output.
- This makes the proxy easier to monitor during development.

### 3.3.4 Summary

SaltOS4’s service worker acts as a smart, low-level proxy that enables offline functionality and efficient request handling. It ensures smooth interaction with the backend while minimizing the impact of network failures or latency.

+Authentication++

SaltOS4 uses JavaScript modules to handle login and authentication on the client side. The core logic is implemented in ‘web/js/auth.js‘, while the login form logic is defined in ‘apps/users/js/login.js‘.

### 3.3.5 Core Auth Module

The ‘saltos.authenticate‘ module provides these functions:

- ‘authtoken(user, pass)‘: Sends credentials to the backend (‘auth/login‘). If successful, stores the token using ‘saltos.token.set(...)‘.
- ‘checktoken()‘: Validates the current token by calling ‘auth/check‘.
- ‘deauthtoken()‘: Logs out the user and removes the token (‘auth/logout‘).
- ‘authupdate(old, new, renew)‘: Changes the user’s password (‘auth/update‘).

These functions internally use ‘saltos.app.ajax(...)‘, which wraps ‘saltos.core.ajax(...)‘ for simplified usage.

### 3.3.6 Login Workflow

The ‘saltos.login.authenticate‘ function handles the login form:

- Validates that required fields are filled.
- Retrieves ‘user‘ and ‘pass‘ from the form.
- Calls ‘saltos.authenticate.authtoken(...)‘.
- On failure, shows an error with ‘saltos.app.toast(...)‘.
- On success:
- Redirects to ‘app/dashboard‘ if login was accessed directly.
- Triggers the global event ‘saltos.app.login‘.

This flow enables seamless login via the web UI while leveraging the same token-based backend used for API and CLI access.

+Build System++

SaltOS4 provides two build modes for the frontend: production and development. These modes are designed to balance efficiency and ease of debugging, and they are automatically generated using the ‘make web‘ and ‘make devel‘ targets in the project’s ‘Makefile‘.

### 3.3.7 Overview of Modes

- **Production Mode (‘make web‘)**:
  - Combines and minifies all JavaScript into ‘index.js‘.
  - Combines the service worker proxy and MD5 loader into ‘proxy.js‘.
  - Generates a minified ‘index.htm‘ with ‘integrity‘ hashes and MD5-based cache busting.
  - Loads minimal assets, optimized for performance and security.
- **Development Mode (‘make devel‘)**:
  - Loads all source files individually from the ‘js/‘ folder.
  - Skips minification, hashing, and integrity checks.
  - Generates an ‘index.htm‘ with expanded script references for direct debugging.
  - ‘proxy.js‘ becomes a lightweight loader with ‘importScripts(...)‘.

### 3.3.8 Build Process with Makefile

- ‘make web‘:
  - Uses scripts such as ‘fixpath.php‘, ‘md5sum.php‘, ‘uglifyjs‘, ‘sha384.php‘.
  - Populates ‘index.htm‘ based on the ‘web/htm/index.htm‘ template, inserting hashes and integrity attributes.
- ‘make devel‘:
  - Uses ‘debug.php‘ to create a loader that references unminified files directly.
  - Generates a simplified ‘proxy.js‘ pointing to the original sources.

Both modes are based on the same HTML loader template found in ‘web/htm/index.htm‘, which is dynamically populated depending on the mode.

### 3.3.9 Web Directory Overview

- ‘api/‘: symlink to the backend API (‘code/api‘)
- ‘apps/‘: symlink to application code (‘code/apps‘)
- ‘htm/‘: loader templates for generating ‘index.htm‘ and ‘ping.htm‘
- ‘img/‘: interface images and logos
- ‘index.htm‘: main entry point, minified in production, expanded in development
- ‘index.js‘: combined JavaScript (production only)
- ‘index.js.map‘: sourcemap for ‘index.js‘ (production only)
- ‘proxy.js‘: combined service worker or loader depending on mode
- ‘proxy.js.map‘: sourcemap (production only)
- ‘js/‘: source JavaScript code
- ‘lib/‘: third-party libraries

### 3.3.10 Comparison Table

| Feature | Production Mode (‘make web‘) | Development Mode (‘make devel‘) |
|---|---|---|
| Main JS | Combined in ‘index.js‘ | Loaded as individual source files |
| Proxy | Minified ‘proxy.js‘ | Lightweight ‘importScripts(...)‘ loader |
| HTML Loader | Minified ‘index.htm‘ with ‘integrity‘ | Expanded ‘index.htm‘ with raw script tags |
| Integrity Check | Enabled (SRI hashes) | Disabled (empty ‘integrity‘ attributes) |
| Cache Busting | Enabled (MD5 hashes) | Disabled |
| Sourcemaps | Included for debugging | Uses original sources directly |
| Goal | Efficiency and security | Debugging and development flexibility |

# 4 Application System

Applications in SaltOS4 are modular, defined in folders under ‘apps/‘.

Each folder may contain:

- ‘xml/manifest.xml‘: declares the apps
- ‘xml/*.xml‘ or ‘xml/*.yaml‘: application definitions
- ‘php/‘, ‘js/‘, ‘locale/‘: optional logic and translations
- ‘dbschema.xml‘, ‘dbstatic.xml‘: database structure and static content

## 4.1 Manifest Files

The ‘manifest.xml‘ file defines the metadata and registration details for each SaltOS4 application. It is a declaration file that the system reads to know how to display, categorize, and enable the app. Its use makes the system modular and extensible.

Each ‘manifest.xml‘ is usually located at ‘apps/<appname>/xml/‘.

### 4.1.1 General Structure

The general structure of the file is as follows:

```
<root>
    <groups>
        <group code="group_code" name="Group Name" description="Description
            "
                color="color" position="number"/>
    </groups>
    <apps>
        <app id="number" active="1|0" code="app_code" name="App Name"
            description="Description"
              group="group_code" position="number"
              color="color" opacity="0|25|50|75|100" fontsize="1-6"
              table="main_table" field="main_field"
              subtables="alias:table(foreign_key),..."
              has_index="1|0" has_control="1|0" has_version="1|0"
              has_files="1|0" has_notes="1|0" has_log="1|0"
              widgets="widget1,widget2"
              perms="*" allow="1" deny="0"/>
    </apps>
</root>
```

### 4.1.2 Groups (<group>)

- **Identity:**
  - ‘code‘: unique identifier.
  - ‘name‘: display name.
  - ‘description‘: visible description.
- **Visual (optional):**
  - ‘color‘: Bootstrap class (primary, secondary, success, info, warning, danger).
  - ‘position‘: descending order number (higher appears first).

Note: All visual attributes are optional; if not defined, they are not applied.

### 4.1.3 Apps (<app>)

- **Identity:**
  - ‘id‘: unique numeric identifier.
  - ‘active‘: 1 (active), 0 (inactive).
  - ‘code‘: internal code (used in URLs, API, etc.).
  - ‘name‘: display name.
  - ‘description‘: short description.
- **Visual (optional):**
  - ‘group‘: group code it belongs to.
  - ‘position‘: order within the group (higher appears first).
  - ‘color‘: Bootstrap class (same as <group>).
  - ‘opacity‘: 0, 25, 50, 75, 100 → ‘opacity-x‘ class (dashboard only).
  - ‘fontsize‘: 1-6 → ‘fs-x‘ class (dashboard only).
- **Data:**
  - ‘table‘: main associated table.
  - ‘field‘: representative field or computed string.
  - ‘subtables‘: related subtables (alias:table(foreign_key),...)
- **Feature Modules:**
  - ‘has_index‘: enables automatic indexing (creates Mroonga table).
  - ‘has_control‘: enables ownership and sharing control.
  - ‘has_version‘: enables version control (with local blockchain).
  - ‘has_files‘: enables files management.
  - ‘has_notes‘: enables notes management.
  - ‘has_log‘: enables data access logging.
- **Widgets:**
  - ‘widgets‘: list of widgets provided by the app for the dashboard.
- **Permissions:**
  - ‘perms‘: list of permissions (comma-separated or using ‘*‘ as wildcards).
  - ‘allow‘: 1 to force global access.
  - ‘deny‘: 1 to force global denial.

Notes:

All visual attributes are optional; if not defined, they are not applied.

If the permission refers to a permission code without an ‘owner‘ (e.g., ‘main‘, ‘menu‘, ‘create‘), you can simply write the code as is.

For permission codes that support ownership (like ‘list‘, ‘view‘, ‘edit‘, ‘delete‘), you must specify the full code

by combining the permission and the owner (e.g., ‘viewall‘, ‘viewuser‘, ‘viewgroup‘), or use a wildcard (e.g., ‘view*‘) to include all ownership levels.

Example:

- ‘main,create,viewall‘ (valid)
- ‘view‘ (invalid → must be ‘viewall‘, ‘viewuser‘ or ‘viewgroup‘, or use ‘view*‘)

Using wildcards (e.g., ‘view*‘) is common to grant all ownership levels.

### 4.1.4 Permission System

SaltOS4 defines standard permissions in ‘tbl_perms‘:

| Code | Owner | Name |
|---|---|---|
| main | | Main access |
| menu | | Show in menu |
| create | | Create records |
| widget | | Show widget |
| action | | Execute action |
| config | | Configure app |
| help | | Show help |
| info | | Info feature |
| list | user | List only user records |
| list | group | List user and group records |
| list | all | List all records |
| view | user | View details only user records |
| view | group | View details user and group records |
| view | all | View details all records |
| edit | user | Modify only user records |
| edit | group | Modify user and group records |
| edit | all | Modify all records |
| delete | user | Delete only user records |
| delete | group | Delete user and group records |
| delete | all | Delete all records |

‘allow="1"‘ forces global access; ‘deny="1"‘ globally denies access.

Permissions codes with an ‘owner‘ require specifying the permission together with the owner. For example, instead of ‘view‘, use ‘viewall‘, ‘viewgroup‘ or ‘viewuser‘.

Wildcards like ‘view*‘ are supported to include all ownership levels at once.

Simple permission codes (those without owner) like ‘main‘, ‘menu‘, ‘create‘ can be used directly.

### 4.1.5 Feature Modules

The feature modules (‘has_*‘) add extra functionality:

- ‘has_index‘: automatically creates a full-text index table using Mroonga as a storage engine inside MariaDB, enabling high-performance searches without external services like Elasticsearch. This index includes not only the fields of the app’s main table but also any defined subtables, as well as related files

and notes for each record. The system keeps the index automatically updated on each data change, ensuring consistent and up-to-date search results across all linked information.

- 'has_control': enables the ownership and sharing control module, adding metadata to each record to store the creator's 'user_id', 'group_id', and optional lists of additional users and groups with whom the record is shared. It also tracks a creation timestamp. This module allows precise control over who owns and who can access each record at the user and group level. It automatically adds user interface elements to view and manage these permissions directly from the record screens.
- 'has_version': enables automatic version control for the app. Every change to a record is stored as a delta in a dedicated version table, preserving the full history of modifications over time. This version data is secured using a local blockchain to ensure data integrity and prevent tampering. The user interface automatically includes a button on each record's detail view to access and review the version history, showing who made each change, what was changed, and when.
- 'has_log': activates access and query logging for the app, creating a log table that records every read, write, update, or delete operation performed on the app's data. This provides complete traceability of data access for auditing purposes. Like 'has_version', a button is automatically added to the record detail view to allow users to consult the access log for each record, showing who accessed the data, from where, and when.
- 'has_files': automatically adds the SaltOS4 file module to the app. This creates an additional table to store files related to each record in the app's main table. It also automatically updates the user interface to include:
  - a button to upload new files on create and edit screens,
  - a table showing attached files on view and edit screens. This functionality is integrated into the core system so that any app enabling 'has_files' gains built-in file upload, management, and display features without needing extra development.
- 'has_notes': similar to 'has_files', but for notes. It creates a notes table linked to the main table and automatically adds UI elements to allow adding new notes and displaying all notes associated with each record on relevant screens. Like 'has_files', this module is pre-integrated into the core and ready for any app that enables it.

### 4.1.6 Special Cases

- Apps without 'group': e.g., 'login', 'dashboard' (embedded or functional apps).
- Apps without 'table' and 'field': e.g., 'login', 'dashboard', 'certs' (use backend functions).
- Groups defined in different files are globally merged.
- If visual attributes are not defined → no effect (equivalent to null).

### 4.1.7 Example

```
<root>
    <groups>
        <group code="crm" name="CRM" description="Customer Management"
           color="success" position="4"/>
        <group code="sales" name="Sales" description="Sales Management"
           color="info" position="3"/>
    </groups>
    <apps>
        <app id="1" active="1" code="login" name="Login" description="Login
            screen"
```

```
            perms="main" allow="1"/>
        <app id="2" active="1" code="dashboard" name="Dashboard"
            description="Main dashboard"
            perms="main,config,help" allow="1"/>
        <app id="50" active="1" code="customers" name="Customers"
            description="Manage customers"
            group="crm" position="1" color="success" fontsize="1"
            table="app_customers" field="name"
            has_index="1" has_control="1" has_version="1" has_files="1"
                has_notes="1" has_log="1"
            widgets="last_customers,new_customers" perms="*"/>
        <app id="53" active="1" code="invoices" name="Invoices" description
            ="Manage invoices"
            group="sales" position="1" color="info" opacity="50" fontsize
                ="3"
            table="app_invoices" field="IF(is_closed, invoice_code,
                proforma_code)"
            subtables="lines:app_invoices_lines(invoice_id),taxes:
                app_invoices_taxes(invoice_id)"
            has_index="1" has_control="1" has_version="1" has_files="1"
                has_notes="1" has_log="1"
            widgets="last_7_invoices,invoice_total_by_day"
            perms="main,create,edit,viewall,deleteall"/>
    </apps>
</root>
```

### 4.1.8 Usage Notes

- Groups and apps are ordered by ‘position‘ descending; within the same position, alphabetically.
- Visual attributes are optional; if not defined they have no effect and only affect the UI appearance, not backend logic.
- ‘perms‘ supports ‘*‘ and wildcards (e.g., ‘view*‘), the ‘perms‘ attribute controls what actions are available (e.g., ‘main, create, edit, delete, view‘) or can be set to ‘*‘ for full access.
- The system reads all ‘manifest.xml‘ files when refreshing the app registry.
- Disabling an app (‘active=”0”‘) will hide it from UI but preserve its data in the database.
- Subtables help link detailed records like line items or attachments and are used for indexing relationships.
- Apps without ‘group‘ or without ‘table‘/‘field‘ are valid depending on their function (e.g., embedded apps like ‘login‘ or ‘dashboard‘) and may fetch data directly through backend functions instead of database queries.
- Feature modules activated via ‘has_*‘ automatically create and manage their corresponding tables and features.
- ‘has_index‘ enables full-text indexing using Mroonga, integrated as a storage engine in MariaDB for high-performance searches without external services like Elasticsearch.
- ‘has_version‘ enables automatic version control, storing each change as a delta in a version table, secured with a local blockchain to guarantee data integrity.
- ‘has_control‘ adds ownership metadata (user_id, group_id, sharing) and timestamps for each record.
- ‘has_files‘ and ‘has_notes‘ add UI elements and storage for user-uploaded files and notes.

- ‘has_log‘ creates a log table to register access and queries for the app, providing traceability.
- For permissions requiring ownership, you must specify the full code (e.g., ‘viewall‘ instead of ‘view‘), or use a wildcard (e.g., ‘view*‘).

This manifest system makes SaltOS4 highly modular and easy to extend.

## 4.2 YAML-based Apps

SaltOS4 allows simple applications to be defined using YAML. This format is ideal for CRUD-style interfaces with basic forms and lists. Below are real-world examples used in the system.

### 4.2.1 tokenslog.yaml

```
app: tokenslog
require: apps/common/php/default.php
template: apps/common/xml/default.xml
indent: true
screen: type2
col_class: col-md-6 mb-3
dropdown: false
list:
    # [id, type, label]
    - [user_id, select, User]
    - [created_at, text, Created]
    - [token, text, Token]
    - [active, boolean, Active]
form:
    # [id, type, label]
    - [active, switch, Active]
    - newline
    - [user_id, select, User]
    - [created_at, datetime, Created]
    - [updated_at, datetime, Updated]
    - [remote_addr, text, Remote Addres]
    - [user_agent, text, User Agent]
    - [token, text, Token]
    - [expires_at, datetime, Expires]
select:
    # [id, table, optional field]
    - [user_id, tbl_users]
attr:
    # field:
    #     attr: value
    user_agent:
        col_class: col-md-12 mb-3
```

### 4.2.2 configlog.yaml

```
app: configlog
require: apps/common/php/default.php
template: apps/common/xml/default.xml
indent: true
```

```
screen: type2
col_class: col-md-6 mb-3
dropdown: false
list:
    # [id, type, label]
    - [user_id, select, User]
    - [key, text, Key]
form:
    # [id, type, label]
    - [user_id, select, User]
    - [key, text, Key]
    - [val, codemirror, Value]
select:
    # [id, table, optional field]
    - [user_id, tbl_users]
attr:
    # field:
    #     attr: value
    key:
        required: true
        autofocus: true
    val:
        required: true
        mode: json
        indent: true
        col_class: col-md-12 mb-3
        height: 5em
```

### 4.2.3 taxes.yaml

```
app: taxes
require: apps/common/php/default.php
template: apps/common/xml/default.xml
indent: true
screen: type5
col_class: col-md-6 mb-3
dropdown: false
list:
    # [id, type, label]
    - [name, text, Name]
    - [description, text, Description]
    - [value, text, Value]
    - [active, boolean, Active]
    - [default, boolean, Default]
form:
    # [id, type, label]
    - [active, switch, Active]
    - [default, switch, Default]
    - [name, text, Name]
    - [value, float, Value]
    - [description, textarea, Description]
attr:
    # field:
    #     attr: value
```

```
    name:
        required: true
        autofocus: true
    description:
        col_class: col-md-12 mb-3
        height: 5em
```

These examples demonstrate the simplicity of defining apps in YAML.

### 4.2.4 Advanced YAML file documentation

In SaltOS4, a YAML file defines an application's interface and configuration declaratively. It serves as an intermediate specification that the system transforms into an XML file, which is the format actually processed by SaltOS4.

When a YAML-defined app is loaded, SaltOS4 uses the 'require' and 'template' keys to generate the XML dynamically. This generated XML is stored in the cache directory and includes a comment line like:

```
<!-- source: apps/crm/xml/customers.yaml -->
```

This comment traces the XML back to its original YAML source.

The 'indent' key controls whether indentation is applied when generating the XML output. The 'col_class' key sets a default Bootstrap class for all form fields in the app, unless overridden at the field level via 'attr'.

The 'dropdown' key controls whether action icons are displayed inline or inside a dropdown menu when multiple actions exist for a record.

Lines starting with '#' are comments in YAML and are ignored by the parser.

### 4.2.5 YAML structure

Below is an example YAML structure:

```
app: myapp
require: apps/common/php/default.php
template: apps/common/xml/default.xml
indent: true
screen: type5
col_class: col-md-6 mb-3
dropdown: false
list:
    - [name, text, Name]
    - [active, boolean, Active]
form:
    - [name, text, Name]
    - [description, textarea, Description]
    - [active, switch, Active]
select:
    # [id, table, optional field]
    - [type_id, app_customers_types]
attr:
    name:
        required: true
        autofocus: true
```

```
    notes:
        col_class: col-md-12 mb-3
        height: 5em
    city:
        col_class: col-md-3 mb-3
    province:
        col_class: col-md-3 mb-3
    zip:
        col_class: col-md-3 mb-3
    country:
        col_class: col-md-3 mb-3
    code:
        col_class: col-md-3 mb-3
    email:
        col_class: col-md-4 mb-3
    phone:
        col_class: col-md-4 mb-3
    website:
        col_class: col-md-4 mb-3
    type_id:
        col_class: col-md-3 mb-3
```

### 4.2.6 Explanation of YAML fields

- ‘app‘: internal code (unique identifier) for the app
- ‘require‘: PHP file to include as backend logic
- ‘template‘: XML template used as a base for the interface
- ‘indent‘: whether to apply indentation (boolean)
- ‘screen‘: type of layout to use (type1, type2, type3, type4, type5)
- ‘col_class‘: default Bootstrap class applied to all form fields
- ‘dropdown‘: whether to force inline icons or use a dropdown menu
- ‘list‘: defines the list columns; each entry is field, type,
- ‘form‘: defines the form fields; each entry is field, type,
- ‘select‘: defines relationships between fields and external tables
- ‘attr‘: specifies additional attributes per field inside ‘form‘

### 4.2.7 List and Form schema

Both ‘list‘ and ‘form‘ sections use the same schema per field: ‘id, type,‘.

For ‘list‘, supported ‘type‘ values are:

- ‘text‘: displays plain text
- ‘select‘: resolves the field by querying a related table and displaying the linked label
- ‘boolean‘: shows a green or red icon (true if value != 0, false if value == 0)
- ‘hastext‘: like ‘boolean‘, but true if the field contains text, false if empty

For ‘form‘, supported ‘type‘ values are:

- ‘text‘, ‘date‘, ‘time‘, ‘datetime‘, ‘checkbox‘, ‘switch‘, ‘integer‘, ‘float‘, ‘color‘, ‘textarea‘, ‘ckeditor‘, ‘codemirror‘, ‘select‘, ‘multiselect‘, ‘newline‘

All these types correspond to widgets provided by SaltOS4, **except ‘newline‘**, which is a special case. The ‘newline‘ type injects a ‘ ‘ with ‘col_class=”col-12”‘ in the XML, effectively creating a line break in the form layout for grouping fields visually.

### 4.2.8 Select section

The ‘select‘ section allows mapping form fields to other tables. Each entry is ‘id, table,‘. If ‘optional_field‘ is omitted, SaltOS4 tries to auto-resolve the master field from the target app’s manifest.

### 4.2.9 Attr section

The ‘attr‘ section adds custom attributes per form field in the XML widget, such as ‘required=”true”‘, ‘autofocus=”true”‘, ‘height=”5em”‘, or overrides like ‘col_class‘.

### 4.2.10 Summary

The YAML file defines:

- Declarative structure for the app’s form and list
- Default and override attributes for fields
- Table relationships via ‘select‘
- UI behaviors like dropdowns and line breaks

SaltOS4 transforms this YAML into a complete XML interface definition, enabling fast and flexible app development.

## 4.3 XML-based Complex Apps

SaltOS4 also allows full custom apps defined in XML, often used for more complex business logic and interface customization.

### 4.3.1 invoices.xml

The following is a small example illustrating the structure of an application XML file:

```
<main>
  <screen>type5</screen>
  <title>Invoices</title>
  <navbar>
    .
    .
    .
  </navbar>
  .
  .
```

```
  .
</main>
<list>
  .
  .
  .
</list>
<form>
  .
  .
  .
</form>
.
.
.
```

This file contains the follow important nodes:

- '<main>': Defines a call to app/customers/main that returns the screen type, literals, and the navbar specification
- '<list default="true">': Defines a call to app/customers/list that returns a cache load from app/customers/list/cache
- '<list id="cache">': Defines a call to app/customers/list/cache that returns the interface of a list screen
- '<list id="data">': Defines a call to app/customers/list/data that returns the data of a list
- '<_form>': This defines a form, there is no way to access it externally, it is for internal use
- '<_data require="php/lib/log.php" eval="true">': This defines a data block of a detail view, not accessible externally, intended for internal use
- '<create>': Defines a call to app/customers/create that returns a cache load from app/customers/create/cache
- '<create id="cache">': Defines a call to app/customers/create/cache that returns the interface of a creation screen
- '<create id="insert">': Defines a call to app/customers/insert that allows inserting a record and returns the status
- '<view>': Defines a call to app/customers/view that returns a cache load from app/customers/view/-cache
- '<view id="cache">': Defines a call to app/customers/view/cache that returns the interface of a creation screen
- '<edit>': Defines a call to app/customers/edit that returns a cache load from app/customers/edit/-cache
- '<edit id="cache">': Defines a call to app/customers/edit/cache that returns the interface of a creation screen
- '<edit id="update">': Defines a call to app/customers/update that allows updating a record and returns the status
- '<delete>': Defines a call to app/customers/delete that allows deleting a record and returns the status
- '<action id="xxx">': Defines a call to app/customers/xxx that allows to execute the xxx action for this application

## 4.4 Available Layouts Type1 To Type5

The interface engine (driver.js) in SaltOS4 supports multiple layout types depending on the application's needs:

- type1: Simple layout with a single zone (#one). Each action opens a new view.
- type2: Two zones — #one for the list and #two for details and attachments.
- type3: Three zones — #one, #two, and #three — allowing all views at once.
- type4: Extends type1. Adds #two as a modal to show details while keeping #one as the main area.
- type5: Extends type2. Shows details in #two and uses #three as a modal for attachments.

Mobile adaptation:

If an app is configured with type2, type3, type4 or type5 and the window.innerWidth is less than 1200 pixels, the system automatically switches to type1. This improves usability on mobile devices by simplifying the workflow and avoiding cluttered interfaces.

## 4.5 Data Model using dbschema.xml

Each application in SaltOS4 can define its own database schema through a 'dbschema.xml' file located in the 'xml/' folder. This file describes the required tables, fields, types, indexes, and relationships.

There is also a central file ('api/xml/dbschema.xml') that defines the **core system tables** used by SaltOS4, including:

- tbl_apps, tbl_perms, tbl_users_apps_perms, tbl_groups_apps_perms: permissions per user and group.
- tbl_uploads: temporary uploaded files.
- tbl_cron: log of scheduled tasks.
- tbl_push: push notifications.
- tbl_trash: deleted records.

### 4.5.1 Structure of dbschema.xml

A typical 'dbschema.xml' file includes:

- '<table>': Defines a database table.
  - 'name': Table name.
- '<fields>': Contains '<field>' entries:
  - 'name': Column name.
  - 'type': SQL type (supports MySQL and SQLite variants).
  - 'pkey': Marks the primary key if 'true'.
  - 'fkey': Defines a foreign key reference.
- '<indexes>' (optional): Contains '<index>' entries listing:
  - 'fields': Comma-separated list of fields in the index.

### 4.5.2 Example

Example (trimmed here for brevity, see full XML files in repository):

```
<table name="app_taxes">
    <fields>
        <field name="id" type="/*MYSQL INT(11) *//*SQLITE INTEGER */" pkey
            ="true"/>
        <field name="name" type="VARCHAR(255)"/>
        <field name="description" type="TEXT"/>
        <field name="value" type="FLOAT"/>
        <field name="active" type="INT(11)"/>
        <field name="default" type="INT(11)"/>
    </fields>
</table>
```

A taxes table might include:

- Primary key id (integer).
- Fields like name (varchar), description (text), value (float), active (integer), and default (integer).
- No special indexes beyond the primary key (unless defined explicitly).

### 4.5.3 Automatic Schema Management

SaltOS4 automatically reads the ‘dbschema.xml‘ definitions and compares them against the live database. If it detects changes (new fields, indexes, tables), it applies the required SQL commands **automatically** to synchronize the schema.

This means:

- Developers do **not** need to write migration scripts.
- Admins do **not** need external tools (like phpMyAdmin).
- Schema updates happen transparently during deployment or app installation.

### 4.5.4 System-Wide and App-Specific Schemas

- The **system-wide schema** (‘api/xml/dbschema.xml‘) defines shared tables.
- Each **app-specific schema** (‘apps/<app>/xml/dbschema.xml‘) defines tables used only by that app.

The system merges both to build the complete data model.

### 4.5.5 Benefits of dbschema.xml

- Guarantees consistency across MySQL and SQLite setups.
- Acts as the single source of truth for the data structure.
- Enables automatic upgrades without manual SQL.
- Simplifies building custom apps with tailored tables.
- Prevents schema drift between environments.

### 4.5.6 Complement: dbstatic.xml

SaltOS4 also uses a ‘dbstatic.xml‘ file to define **static data** (preloaded values) for:

- Default permissions.
- System users and groups.
- Registered apps.
- Essential relationships.

Together, ‘dbschema.xml‘ and ‘dbstatic.xml‘ allow SaltOS4 to fully bootstrap a new installation, ensuring it is immediately usable after setup.

## 4.6 Static Data using dbstatic.xml

The file ‘dbstatic.xml‘ defines static records that are inserted into the database during system initialization. These values are essential for SaltOS4 to operate correctly and include:

- Default permissions and ownership levels
- Core user and group records
- Registered applications in the system
- Permission-to-application assignments
- Initial permission grants for the main user

These static values are inserted during system initialization and are critical to bootstrapping SaltOS4 correctly. They are also used by the ‘make setup‘ and ‘make setupinstall‘ targets to preload essential entities.

Example from ‘api/xml/dbstatic.xml‘:

- Table ‘tbl_perms‘: defines available permissions like ‘main‘, ‘create‘, ‘widget‘, ‘action‘, etc., with optional ownership levels (‘user‘, ‘group‘, ‘all‘).
- Other tables likely define rows for default users, groups, apps, or app-permission mappings, though they are not visible in the excerpt shown.

Each ‘<row>‘ inside ‘<table>‘ specifies a record to insert, using attributes like ‘id‘, ‘code‘, ‘name‘, ‘owner‘, and ‘active‘.

This static data ensures consistent behavior across installations and is critical to bootstrapping the system correctly.

## 4.7 How to Add a New App

Adding a new app in SaltOS4 involves several steps that combine both backend and frontend configuration.

### 4.7.1 Create the App Folder

Inside code/apps/, create a new folder with the app name, e.g. myapp/. This folder will contain:

- php/: backend PHP logic (optional).
- js/: frontend JavaScript logic (optional).

- xml/: configuration files like manifest.xml and optionally dbschema.xml, dbstatic.xml.
- locale/: translations (optional).

### 4.7.2 Define the Manifest

Create xml/manifest.xml inside the app folder. This file declares the app to the system, specifying:

- Unique id (check unused IDs in existing manifests).
- code, name, description.
- Associated group (like crm, sales, etc).
- Main table, field, and optional subtables.
- Flags like has_index, has_control, has_version, etc.
- Required permissions (perms).

Example:

```
<root>
    <apps>
        <app id="100" active="1" group="crm" code="new" name="New App"
           description="My new application" table="app_new" field="name"
           perms="main,menu,create,list,view,edit,delete"/>
    </apps>
</root>
```

### 4.7.3 Define the Database Schema

Create xml/dbschema.xml to declare the app's database tables. SaltOS4 will read and apply this schema automatically.

Example for a simple table:

```
<root>
    <tables>
        <table name="app_myapp">
            <fields>
                <field name="id" type="INTEGER" pkey="true"/>
                <field name="name" type="VARCHAR(255)"/>
                <field name="description" type="TEXT"/>
                <field name="active" type="INT(11)"/>
            </fields>
        </table>
    </tables>
</root>
```

### 4.7.4 Define the Static Data (Optional)

Use xml/dbstatic.xml if the app needs initial records (like predefined types or statuses).

### 4.7.5 Add the Application Definition

For a YAML-based app, create a file like xml/myapp.yaml to define the screens, forms, and lists.

Example:

```
app: myapp
require: apps/common/php/default.php
template: apps/common/xml/default.xml
indent: true
screen: type5
col_class: col-md-6 mb-3
dropdown: false
list:
    - [name, text, Name]
    - [active, boolean, Active]
form:
    - [name, text, Name]
    - [description, textarea, Description]
    - [active, switch, Active]
attr:
    name:
        required: true
        autofocus: true
    description:
        required: true
```

### 4.7.6 Ensure Symbolic Links

SaltOS4 uses symbolic links to map directories:

- In ‘api/‘, link ‘apps‘ → ‘../code/apps‘ and ‘data‘ → ‘../code/data‘.
- In ‘web/‘, link ‘api‘ → ‘../code/api‘ and ‘apps‘ → ‘../code/apps‘.

This allows the frontend and backend to access shared app resources consistently.

Summary:

- The backend (api/) accesses apps/ and data/ via local symlinks.
- The frontend (web/) accesses the backend (api/) and apps/ via local symlinks.
- This setup ensures both frontend and backend can reach the same app and data resources.

### 4.7.7 Run Database Setup

Apply the new schema and static data:

```
make setup
```

### 4.7.8 Assign Permissions

Grant access to the new app for users or groups using the admin interface or database configuration.

### 4.7.9 Test the App

Navigate to the app from the SaltOS4 interface and check that:

- Lists and forms load correctly.
- Data can be inserted, updated, and deleted.
- Permissions are enforced.

This structured process ensures the new app integrates smoothly with the SaltOS4 framework and follows its conventions.

# 5 Makefile Overview

This ‘Makefile‘ automates building, testing, documentation, and setup tasks in SaltOS4.

## 5.1 Build Targets

SaltOS4 includes several ‘make‘ targets to manage the build process for both production and development environments. These targets automate the generation of optimized assets, cleanup of temporary files, and preparation of debug-friendly output.

- ‘make web‘: Builds and minifies production assets:
  - Combines and compresses CSS/JS
  - Uses ‘fixpath.php‘, ‘md5sum.php‘, ‘uglifyjs‘, ‘minify‘, ‘sha384.php‘
  - Handles app-specific JS from ‘apps/*/js/*.js‘
  - Generates the ‘proxy.js‘ script with source maps
- ‘make devel‘: Prepares a development environment with unminified assets using ‘debug.php‘.
- ‘make clean‘: Deletes generated files:
  - Minified JS, CSS, maps, HTML, ‘proxy.js‘, and per-app compiled assets

## 5.2 Documentation

The ‘make docs‘ target is used to automatically generate documentation for multiple source areas. It leverages PHP scripts to extract comments and convert ‘.t2t‘ files into both PDF and HTML formats.

You can control which documentation files are processed using the ‘file‘ parameter. If no value is passed, all sections are generated by default.

Examples:

- ‘make docs‘ → generates everything
- ‘make docs file=api‘ → generates only API documentation
- ‘make docs file=api,web‘ → generates both API and web documentation

Supported sections:

- ‘api‘: generates the backend doc ‘docs/api.t2t‘ from ‘code/api/php‘

- ‘web‘: generates the fronend doc ‘docs/web.t2t‘ from ‘code/web/js‘
- ‘apps‘: generates the applications doc ‘docs/apps.t2t‘ from ‘code/apps/*/php‘ and ‘code/apps/*/js‘
- ‘utest‘: generates the php unit test doc ‘docs/utest.t2t‘ from ‘utest/‘
- ‘ujest‘: generates the javascript unit test ‘docs/ujest.t2t‘ from ‘ujest/‘
- ‘devel‘: updates de developer manual ‘docs/devel.t2t‘ (version/date) and regenerates its output

This logic is implemented using pattern matching via ‘findstring‘ in the Makefile, allowing comma-separated values to select multiple targets at once.

## 5.3 Testing

SaltOS4 integrates various testing mechanisms to maintain code quality across both PHP and JavaScript components. The ‘make‘ system provides convenient commands for running static analysis, unit tests, and coverage reports. Below are the available targets:

- ‘make test‘: Runs:
  - ‘phpcs‘, ‘php -l‘, ‘phpstan‘ on PHP files
  - ‘jscs‘, ‘node -c‘ on JS files
  - Accepts variable ‘file=...‘, ‘file=all‘, or none (uses SVN diff)
- ‘make utest‘: Runs PHPUnit with optional filtering by file
- ‘make ujest‘: Runs JS tests with Jest:
  - Cleans diff snapshots and temp coverage
  - Supports full or filtered test runs
  - Generates coverage report

## 5.4 Environment Checks

SaltOS4 includes a helper target to verify that the environment is correctly set up. This check ensures required directories, symbolic links, and external tools are available before running builds, tests, or development tasks.

- ‘make check‘: Verifies required folders and system commands:
  - Symbolic links: ‘api/data‘, ‘web/apps‘, etc.
  - Commands: ‘php‘, ‘node‘, ‘jest‘, ‘phpunit‘, ‘uglifyjs‘, ‘txt2tags‘, etc.

## 5.5 Dependency Management

‘make libs‘ executes ‘scripts/checklibs.php‘, which checks for new versions of required PHP and JavaScript libraries, as well as other external software used by the system. This mechanism helps monitor version updates and ensures that all dependencies remain current.

Libraries can be integrated through various methods depending on how they are distributed — for example:

- Via ‘wget‘ from a direct URL
- Via ‘npm‘

- Via Composer for PHP packages

The monitoring is based on the file ‘checklibs.txt‘, where each line defines how to retrieve and compare the current version of a library or tool.

Example line from ‘checklibs.txt‘:

```
phpmailer|https://github.com/PHPMailer/PHPMailer/tags.atom|<title>|
    PHRpdGxlPlBIUE1haWxlciA2LjkuMzwvdGl0bGU+
```

As you can see, this line consists of four fields, using pipes (|) as separators:

- The name of the item (library, command, etc.), used as an identifier
- The URL to fetch the latest version information (can be XML, JSON, or plain HTML)
- A regular expression pattern to match the relevant line that announces the latest version
- A base64-encoded string representing the last matched content, used to detect changes

This tells ‘checklibs.php‘ to:

- Retrieve the GitHub tags feed for PHPMailer in Atom (XML) format
- Match the line containing the version information using the regex
- Compare the base64-encoded result with the previously recorded one to determine if an update is available

This system is useful for tracking updates manually without relying on automatic dependency managers.

## 5.6 Code Statistics

‘make cloc‘ uses ‘cloc‘ to count lines of code excluding minified and ignored files, you can see an example here:

```
-------------------------------------------------------------------------------
Language                     files          blank        comment           code
-------------------------------------------------------------------------------
PHP                            302           3809          18565          23321
JavaScript                      44           1040           6596          10536
XML                             47            312           1258           4566
YAML                            15             14            355            450
Text                             4             13              0             93
Bourne Shell                     1              5              0             36
HTML                             1              2             24             15
JSON                             1              0              0             12
-------------------------------------------------------------------------------
SUM:                           415           5195          26798          39029
-------------------------------------------------------------------------------
```

## 5.7 System Setup

These targets initialize or reset the SaltOS4 environment, including database setup and cleanup of data directories. They allow switching between MariaDB and SQLite, and can be used to fully reconfigure the system during development or deployment.

- ‘make setup‘: Initializes SaltOS4 system

- ‘make setupmysql‘: Sets up all applications using MariaDB
- ‘make setupsqlite‘: Sets up all applications using SQLite
- ‘make setupinstall‘: Combines cleaning + full MySQL + SQLite setup
- ‘make setupclean‘: Cleans data directories and resets the database

## 5.8 System Actions

These targets trigger internal maintenance operations in SaltOS4, such as garbage collection, indexing, integrity validation, and scheduled task execution. They help keep the system optimized and consistent.

- ‘make gc‘: Launches garbage collection
- ‘make indexing‘: Performs indexing operations
- ‘make integrity‘: Runs integrity checks
- ‘make cron‘: Executes scheduled tasks (cron)

## 5.9 System langs

This rule runs the ‘scripts/checklangs.py‘ script to validate the integrity of all translation files across languages and groups.

To run the integrity check, use:

```
make langs
```

This will:

- Analyze all ‘manifest.xml‘, ‘.xml‘, ‘.yaml‘, and ‘.js‘ files in every app group
- Load all ‘messages.yaml‘ files from ‘api/locale/‘ and ‘apps/<group>/locale/‘
- Detect duplicate translation keys:
  - Within the same file
  - Between the global and any app group
  - Between two groups (only with ‘–strict‘)
- Report any overlapping definitions that may need cleanup

To perform additional per-language analysis manually, you can call the script with options:

- ‘–lang=<lang>‘: language to check (e.g. ‘ca_ES‘)
- ‘–filter=missing|present|missing_but_in_other_group‘
- ‘–group=<group>‘: restrict to a specific app group
- ‘–csv=<file.csv>‘: export results to a CSV file
- ‘–strict‘: detect duplicate keys across app groups

Examples:

Validate global integrity across all languages:

```
make langs
```

Run language-specific analysis manually:

```
python scripts/checklangs.py --lang ca_ES --group=crm --filter=missing
```

Strict duplicate detection between groups:

```
python scripts/checklangs.py --strict
```

## 5.10 Script Directory Overview

This section describes the purpose of each script and configuration file in the ‘scripts/‘ directory.

All descriptions below are based on the actual content of each file.

- ‘checklangs.py‘: Scans XML/YAML/JS files for translation keys and detects missing or duplicated strings across apps.
- ‘checklibs.php‘: Validates the current versions of required libraries by parsing ‘checklibs.txt‘, performing curl requests, and comparing base64-encoded version strings. Updates the file if needed.
- ‘checklibs.txt‘: Contains a list of required libraries, with their URLs and expected version tags encoded in base64.
- ‘debug.php‘: Generates a debug-friendly version of the frontend using ‘debug.php‘.
- ‘fixpath.php‘: Adjusts file paths in generated HTML/JS/CSS to match the deployment structure.
- ‘jest.config.js‘: Configuration file for running JavaScript unit tests with Jest.
- ‘jest_coverage.php‘: Processes Jest coverage reports and outputs them in a readable format.
- ‘jest_tester.php‘: Parses the layout structure from ‘apps/tester/xml/tester.xml‘, checks for duplicate widget tags, and exports the data as ‘/tmp/tester.json‘.
- ‘jscs.json‘: Defines JavaScript coding standards used by the JSCS code style checker.
- ‘make_bootswatch.php‘: Cleans up Bootswatch CSS files by removing any external ‘@import‘ statements.
- ‘make_instance.sh‘: Creates a new runnable SaltOS4 instance by linking ‘.htaccess‘, preparing folders like ‘data/‘ and ‘tmp/‘, and applying permissions.
- ‘makehtml.php‘: Converts a ‘.t2t‘ documentation file into an HTML file using ‘txt2tags‘.
- ‘maket2t.php‘: Scans a source directory and extracts ‘/** ... */‘ comments from each file to generate a ‘.t2t‘ documentation file.
- ‘makepdf.php‘: Converts a ‘.t2t‘ documentation file into a ‘.pdf‘ using LaTeX.
- ‘md5sum.php‘: Generates an MD5 checksum of one or more files.
- ‘migrate_v3_to_v4.php‘: Migrates a SaltOS system from version 3 to version 4, adapting data and file structure.
- ‘movelangs.py‘: Moves selected translation keys from a group-specific messages.yaml to the global one without altering file structure.
- ‘phpcs.xml‘: Configuration file for PHP_CodeSniffer to enforce PHP coding standards.
- ‘phpstan.neon‘: Configuration file for PHPStan, specifying analysis rules and paths for static code analysis.

- ‘phpunit.xml‘: Configuration file for PHPUnit, specifying test directories, filters, and bootstrap files.
- ‘sha384.php‘: Calculates SHA-384 hashes for files to use in Subresource Integrity (SRI) attributes in HTML.
- ‘updatet2t.php‘: Updates the second and third lines of a ‘.t2t‘ file (used to update the version and date of ‘devel.t2t‘).