Devel Documentation

SaltOS 4.0 r1931

April 2025

Contents

1 Introduction **4**

1.1 Advanced System Features . . . 4

1.2 Project Structure . . . 4

1.3 Code Directory . . . 5

1.4 Data Directory . . . 5

2 Backend Structure (api/) **5**

2.1 Autoloaded Modules (‘php/autoload/‘) . . . 5

2.2 Database Drivers (‘php/database/‘) . . . 6

2.3 Libraries (‘php/lib/‘) . . . 7

2.4 Action Modules (‘php/action/‘) . . . 8

2.5 Other API Components . . . 8

2.6 API Access . . . 9

2.6.1 Web server access (Apache, Nginx, Cherokee) . . . 9

2.6.2 Command-line access (CLI) . . . 9

2.7 API Authentication . . . 9

2.7.1 HTTP Access . . . 10

2.7.2 CLI Access . . . 10

3 Frontend Structure (web/) **10**

3.1 Frontend Deployment . . . 11

3.2 Offline Support (Service Worker) . . . 11

3.3 Proxy Service Worker Internals . . . 12

3.4 Frontend Authentication . . . 13

3.4.1 Core Auth Module . . . 13

3.4.2 Login Workflow . . . 13

3.5 Frontend Build System . . . 13

3.5.1 Overview of Modes . . . 13

3.5.2 Build Process with Makefile . . . 14

4 Application System **15**

4.1 YAML-based Apps . . . 15

4.1.1 tokenslog.yaml . . . 15

4.1.2 configlog.yaml . . . 16
4.1.3 types.yaml . . . 16
4.2 XML-based Complex Apps . . . 17
4.2.1 customers.xml . . . 17
4.3 Available Layouts Type1 To Type5 . . . 18
4.4 Data Model using dbschema.xml . . . 18
4.5 Static Data using dbstatic.xml . . . 18

5 Makefile Overview **19**
5.1 Build Targets . . . 19
5.2 Documentation . . . 19
5.3 Testing . . . 20
5.4 Environment Checks . . . 20
5.5 Dependency Management . . . 20
5.6 Code Statistics . . . 21
5.7 System Setup . . . 22
5.8 System Actions . . . 22
5.9 Script Directory Overview . . . 22

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 Structure (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/ping.php‘: * Ping action
- ‘action/push.php‘: * Garbage Collector action
- ‘action/setup.php‘: * DB Schema action
- ‘action/upload.php‘: * Add files action

2.5 Other API Components

- ‘index.php‘: entry point that loads autoload and then ‘zindex.php‘
- ‘img/‘: SaltOS logos and related branding
- ‘locale/‘: multilingual resources (‘.yaml‘, ‘.odt‘, ‘.pdf‘)
- ‘xml/‘: base configuration
  - ‘config.xml‘: contains the global system configuration, including general options, paths, preferences, and base parameters that affect all of SaltOS4.
  - ‘cron.xml‘: defines the scheduled tasks that the system must run automatically, such as data cleanup, email sending, or synchronizations.
  - ‘locale.xml‘: specifies the available languages in the system and their codes, allowing the interface localization to be managed.
  - ‘bs_theme.xml‘: defines the visual themes compatible with Bootstrap that the system can use to customize the interface appearance.
  - ‘css_theme.xml‘: contains additional visual style definitions, such as colors, fonts, and CSS rules to adapt the system’s look and feel.
  - ‘dbschema.xml‘: describes the general database schema — tables, columns, indexes, and relationships required for the overall functioning of SaltOS4.
  - ‘dbstatic.xml‘: includes static data that must be inserted into certain tables during system initial-

ization, such as user types, default values, or base configurations.

2.6 API Access

SaltOS4 supports access via HTTP or CLI.

2.6.1 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/...‘

2.6.2 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‘

2.7 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.

2.7.1 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.

2.7.2 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.

3 Frontend Structure (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 Frontend 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

3.2 Offline Support (Service Worker)

SaltOS4 includes a service worker defined in ‘js/proxy.js‘ that transparently enables offline support and network optimization.

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 with the core AJAX layer (‘core.js‘) without requiring changes in application logic.
- Logs operations (network, cache, queue, error) using visual debug tools.
- Handles smart caching and fallback behavior depending on the request type.

This proxy mechanism ensures a responsive and resilient user experience even under unreliable network conditions.

3.3 Proxy Service Worker Internals

The file ‘js/proxy.js‘ implements the service worker used by SaltOS4 to act as a network proxy for frontend requests. It includes debugging utilities, a request handler, and caching logic.

Structure Overview

The script is written in strict mode and organized as follows:

- **Banner and License Header**: ASCII logo and GPL license.
- **‘console_log(message)‘**:
  - Sends debug messages to all connected clients.
  - Used instead of ‘console.log()‘ because some browsers suppress SW logs.
- **‘debug(action, url, type, duration, size)‘**:
  - Prepares styled debug output showing request type (‘network‘, ‘cache‘, ‘queue‘, ‘error‘), duration, and response size.
    The actual request handling logic (not fully shown here) includes:
- ‘fetch‘ event listener:
  - Intercepts all network requests.
  - Uses headers to determine cache strategy or fallback behavior.
  - Responds from cache or fetches from the network, optionally logging each action.
    == Cache Strategies ==
    Although not all logic is visible in the current excerpt, the structure suggests support for:
- Smart caching based on headers or resource types.
- Differentiated handling of errors, including timeouts or unreachable hosts.
- Possible queuing or offline fallback for POST/PUT operations.

  == Debugging Support ==

  Service worker debug output is enhanced via:
- ‘console_log(...)‘: forwards logs to browser clients.
- ‘debug(...)‘: shows color-coded summaries in the browser console, including request method, duration, and source (cache/network).

  == Summary ==

  The proxy service worker plays a key role in SaltOS4’s offline capabilities and performance optimization. It acts intelligently based on request metadata and provides a robust debugging interface even when ‘console.log()‘ is unavailable in service workers.

3.4 Frontend 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‘.

3.4.1 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.

3.4.2 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.

3.5 Frontend 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‘.

3.5.1 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(...)‘.

3.5.2 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.

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‘
- ‘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

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 YAML-based Apps

SaltOS4 supports YAML for fast app declarations using the common template logic.

4.1.1 tokenslog.yaml

```
app: tokenslog
require: apps/common/php/default.php
template: apps/common/xml/default.xml
indent: true
screen: type2
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]
    - [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]
```

4.1.2 configlog.yaml

```
app: configlog
require: apps/common/php/default.php
template: apps/common/xml/default.xml
indent: true
screen: type2
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
```

4.1.3 types.yaml

```
app: types
require: apps/common/php/default.php
template: apps/common/xml/default.xml
indent: true
screen: type5
list:
    # [id, type, label]
    - [name, text, Name]
    - [description, text, Description]
    - [active, boolean, Active]
form:
    # [id, type, label]
    - [active, switch, Active]
    - [name, text, Name]
    - [description, textarea, Description]
attr:
    # field:
    #   attr: value
```

```
    name:
        required: true
        autofocus: true
    description:
        required: true
```

These examples demonstrate the simplicity of defining apps in YAML.

4.2 XML-based Complex Apps

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

4.2.1 customers.xml

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="setup">': Defines a call to app/customers/setup that allows to execute the setup for this application

4.3 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.

4.4 Data Model using dbschema.xml

Each application can define its own database schema through a dbschema.xml file. This file declares the required tables and fields. In addition, there is a central file (api/xml/dbschema.xml) that contains the shared tables used system-wide:

- tbl_apps, tbl_perms, tbl_users_apps_perms, tbl_groups_apps_perms: permission control per user and group.
- tbl_uploads: temporarily uploaded files not yet assigned to an app.
- tbl_cron: execution log for scheduled tasks.
- tbl_push: push notifications.
- tbl_trash: records of deleted files.

Examples of dbschema.xml in real apps:

- apps/emails/xml/dbschema.xml: defines tables for emails, accounts, and addresses.
- apps/customers/xml/dbschema.xml: structure for customers including fiscal and geographic data.
- apps/invoices/xml/dbschema.xml: invoice headers, line items, and due dates.
- apps/types/xml/dbschema.xml: generic system for type definitions.

4.5 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

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.

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 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.

- ‘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.
- ‘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‘).