

Package ‘zzedc’

December 18, 2025

Type Package

Title Electronic Data Capture System for Clinical Trials

Version 1.0.0

Author Ronald G. Thomas [aut, cre]

Maintainer Ronald G. Thomas <rgthomas@ucsd.edu>

Description A comprehensive 'Shiny' application for electronic data capture (EDC) in clinical trials. Features include secure user authentication, data entry forms, quality control reports, data visualization, and flexible export capabilities. Built with modern 'bslib' components for responsive design and professional appearance. Supports the zcollab framework for clinical research workflows.

License GPL-3 + file LICENSE

Encoding UTF-8

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.3

Depends R (>= 4.0.0), zzedc.validation (>= 1.0.0)

Imports methods, shiny (>= 1.7.0), bslib (>= 0.4.0), bsicons (>= 0.1.0), shinyjs (>= 2.1.0), DT (>= 0.20), ggplot2 (>= 3.4.0), plotly (>= 4.10.0), dplyr (>= 1.0.0), jsonlite (>= 1.7.0), digest (>= 0.6.0), RSQLite (>= 2.2.0), pool (>= 0.1.6), config (>= 0.3.1), lubridate (>= 1.8.0), stringr (>= 1.4.0), httr (>= 1.4.0), shinyalert (>= 3.0.0), googlesheets4 (>= 1.0.0)

Suggests testthat (>= 3.0.0), knitr, rmarkdown, here, yaml, shinyTime, shinyWidgets, shinysignature, haven

URL <https://github.com/rgt47/zzedc>

BugReports <https://github.com/rgt47/zzedc/issues>

Config/testthat/edition 3

VignetteBuilder knitr

Contents

zzedc-package	3
complete_wizard_setup	4
create_error_display	5
create_launch_script	5

create_paginated_reactive	6
create_pagination_ui	6
create_wizard_config	7
create_wizard_database	7
create_wizard_directories	8
detect_setup_status	8
enable_session_timeout	9
error_response	9
execute_init_setup	10
export_audit_log	10
export_to_file	11
filter_data_by_search	11
generate_db_key	12
generate_export_filename	12
generate_security_salt	13
get_encryption_key	13
get_encryption_key_from_aws_kms	14
get_instrument_field	15
get_page_summary	15
get_setup_instructions	16
handle_error	16
import_instrument	17
init	18
init_audit_log	19
init_from_config	19
init_interactive	20
init_session_timeout	20
is_configured	21
launch_setup_if_needed	21
launch_zzedc	22
list_available_instruments	23
load_instrument_template	23
log_audit_event	24
log_export_event	25
notify_if_invalid	25
paginate_data	26
prepare_all_files_export	27
prepare_edc_export	27
prepare_export_data	28
prepare_reports_export	29
prepare_sample_export	29
query_audit_log	30
renderPanel	30
safe_reactive	31
save_validated_form	32
setup_form_validation	32
setup_pagination_observers	33
sort_data	33
success_response	34
test_encryption	34
update_session_activity	35
validate_field_value	35

zzedc-package3

validate_filename

validate_form

validate_instrument_csv

verify_audit_log_integrity

verify_db_key

%||%

Index40

zzedc-package

ZZedc: Electronic Data Capture System for Clinical Trials

Description

The zzedc package provides a comprehensive Shiny application for electronic data capture (EDC) in clinical trials. It features secure authentication, data entry forms, quality control reports, visualization tools, and export capabilities, all built with modern bslib components.

Details

Key Features:

Authentication & Security:

- Role-based user authentication
- Encrypted data storage
- Audit trail capabilities

Data Entry:

- Customizable data entry forms
- Real-time validation
- Progress tracking

Reporting:

- Basic data summaries
- Data quality reports
- Statistical analysis reports

Data Management:

- Interactive data explorer
- Missing data analysis
- Data visualization tools

Export & Integration:

- Multiple export formats (CSV, Excel, JSON, PDF, HTML)
- Batch export capabilities
- Export templates and scheduling

Getting Started:

To launch the EDC application:

```
library(zzedc)
launch_zzedc()
```

The application will open in your default web browser with a modern, responsive interface powered by Bootstrap 5.

Default Credentials:

For testing purposes, the following credentials are available:

- Username: ww, Password: pw
- Username: q, Password: pw
- Username: w, Password: pw

Note: Change these credentials before deploying to production.

Package Dependencies

This package builds on several excellent R packages including shiny, bslib, DT, ggplot2, plotly, and others to provide a comprehensive EDC solution.

Author(s)

Ronald G. Thomas

See Also

Useful links:

- <https://github.com/rgt47/zzedc>
- Report bugs at <https://github.com/rgt47/zzedc/issues>

complete_wizard_setup *Complete Setup Wizard Orchestration*

Description

Orchestrates all setup steps for initializing a new ZZedc instance

Usage

```
complete_wizard_setup(config_list, base_path = "~/zzedc_instance")
```

Arguments

config_list	Complete configuration from wizard
base_path	Base directory for new installation

Value

List with overall success status and detailed results

create_error_display	<i>Display form validation errors to user</i>
----------------------	---

Description

Creates user-friendly error messages from validation results

Usage

```
create_error_display(validation_result)
```

Arguments

validation_result	
	Result from validate_form()

Value

HTML list of error messages

create_launch_script	<i>Create Launch Script for New ZZedc Instance</i>
----------------------	--

Description

Creates a customized launch script file that users can run to start the application

Usage

```
create_launch_script(config_list, output_path)
```

Arguments

config_list	Configuration from wizard
output_path	Path where launch script will be written

Value

List with success status

`create_paginated_reactive`
Create reactive paginated data

Description

Returns reactive expression that manages paginated data view

Usage

```
create_paginated_reactive(  
  data_source,  
  page_size = 25,  
  search_reactive = NULL,  
  sort_reactive = NULL  
)
```

Arguments

<code>data_source</code>	Reactive data.frame
<code>page_size</code>	Number of rows per page
<code>search_reactive</code>	Optional reactive search term
<code>sort_reactive</code>	Optional reactive sort specification (list with <code>\$by</code> and <code>\$direction</code>)

Value

Reactive expression returning list with paginated data and metadata

`create_pagination_ui` *Create pagination UI controls*

Description

Generates navigation buttons for pagination

Usage

```
create_pagination_ui(pagination, input_id = "data")
```

Arguments

<code>pagination</code>	Pagination info from <code>paginate_data()</code>
<code>input_id</code>	Namespace ID for pagination inputs

Value

HTML div with pagination controls

create_wizard_config	<i>Create Config File from Wizard Configuration</i>
----------------------	---

Description

Creates configuration file with all required application settings

Usage

```
create_wizard_config(config_list, config_path, security_salt)
```

Arguments

config_list	List containing wizard configuration
config_path	Path where config.yml will be written
security_salt	The security salt for hashing

Value

List with success status and messages

create_wizard_database	<i>Create ZZedc Database from Wizard Configuration</i>
------------------------	--

Description

Creates a complete database with all required tables

Usage

```
create_wizard_database(config_list, db_path)
```

Arguments

config_list	List containing wizard configuration (from wizard_state\$system_config)
db_path	Path where database file will be created

Value

List with success status and messages

Examples

```
## Not run:
config <- list(
  study_name = "My Study",
  protocol_id = "PROTO-001",
  admin_username = "admin",
  admin_password = "MyPass123!",
  security_salt = "abc123..."
)
create_wizard_database(config, "~/my_study.db")

## End(Not run)
```

```
create_wizard_directories
```

Create Directories for New ZZedc Instance

Description

Creates the directory structure needed for a new ZZedc installation

Usage

```
create_wizard_directories(base_path)
```

Arguments

base_path Base directory where subdirectories will be created

Value

List with success status

```
detect_setup_status      Detect Configuration Status
```

Description

Comprehensive check of setup status

Usage

```
detect_setup_status()
```

Value

List with status information

`enable_session_timeout`*Enable session timeout monitoring*

Description

Should be called in `server()` function to activate timeout checking. Monitors user inactivity and logs out after configured timeout period.

Usage

```
enable_session_timeout(  
  session,  
  user_input,  
  timeout_config,  
  on_timeout_callback = NULL  
)
```

Arguments

<code>session</code>	Shiny session object
<code>user_input</code>	reactiveValues object containing user session state
<code>timeout_config</code>	List with <code>timeout_minutes</code> (from <code>config\$auth\$session_timeout_minutes</code>)
<code>on_timeout_callback</code>	Function to call when timeout occurs (default: logs out user)

`error_response`*Create standardized error response*

Description

Returns a consistent error response structure.

Usage

```
error_response(message, code = NULL)
```

Arguments

<code>message</code>	Character - error message
<code>code</code>	Character - error code (optional)

Value

List with `success=FALSE` and message

execute_init_setup	<i>Execute Initialization Setup</i>
--------------------	-------------------------------------

Description

Common code for both interactive and config modes

Usage

```
execute_init_setup(config, project_dir = ".")
```

Arguments

config	Configuration list
project_dir	Directory where project will be created

Value

List with setup results

export_audit_log	<i>Export audit log to file</i>
------------------	---------------------------------

Description

Saves audit log to CSV with hash verification included.

Usage

```
export_audit_log(audit_log, filepath, include_verification = TRUE)
```

Arguments

audit_log	reactiveVal or data.frame containing audit records
filepath	Character - path to save audit log
include_verification	Logical - include verification summary?

export_to_file	<i>Export data to file</i>
----------------	----------------------------

Description

Writes export data to specified file format. Supports 9 formats: CSV, XLSX, JSON, SAS, SPSS, STATA, RDS, PDF, HTML

Usage

```
export_to_file(data, filepath, format, options = NULL)
```

Arguments

data	Data to export (data.frame or list)
filepath	Path to write export file
format	Export format (csv, xlsx, json, sas, spss, stata, rds, pdf, html)
options	List of format-specific options

Value

List with success status and file info

filter_data_by_search	<i>Filter data by search term</i>
-----------------------	-----------------------------------

Description

Searches all columns for matching values

Usage

```
filter_data_by_search(data, search_term, columns = NULL)
```

Arguments

data	data.frame to search
search_term	Text to search for (case-insensitive)
columns	Column names to search in (NULL = all columns)

Value

Filtered data.frame

generate_db_key	<i>Generate a random database encryption key</i>
-----------------	--

Description

Creates a cryptographically secure 256-bit random key for SQLCipher database encryption. Returns as a 64-character hexadecimal string.

Usage

```
generate_db_key()
```

Details

Key generation:

- Uses openssl::rand_bytes() for cryptographic security
- 256-bit key = 32 bytes = 64 hex characters
- Never user-provided (best practice: auto-generated)
- Store result in environment variable or AWS Secrets Manager

Value

Character string: 64-hex-character encryption key (256-bit)

Examples

```
## Not run:
key <- generate_db_key()
Sys.setenv(DB_ENCRYPTION_KEY = key)

## End(Not run)
```

generate_export_filename	<i>Generate safe export filename</i>
--------------------------	--------------------------------------

Description

Creates a safe, properly formatted filename for export

Usage

```
generate_export_filename(base_name = NULL, data_source, format)
```

Arguments

base_name	Base filename (user-provided or default)
data_source	Data source identifier
format	Export format (csv, xlsx, json, sas, spss, stata, rds, pdf, html)

Value

Safe filename with extension

```
generate_security_salt
```

Generate Security Salt

Description

Creates a random 32-character salt for password hashing

Usage

```
generate_security_salt()
```

Value

Character string of random salt

```
get_encryption_key
```

Get database encryption key from environment or AWS KMS

Description

Retrieves encryption key with automatic fallback:

1. Try AWS KMS (if credentials and key_id provided)
2. Try environment variable DB_ENCRYPTION_KEY
3. Error if neither available

Usage

```
get_encryption_key(aws_kms_key_id = NULL)
```

Arguments

aws_kms_key_id Character: AWS KMS key ID (optional)

Details

Priority order:

1. AWS KMS (if aws_kms_key_id provided or USE_AWS_KMS=true)
 - Requires paws package
 - Requires AWS credentials (~/.aws/credentials or env vars)
 - Requires AWS IAM permissions for Secrets Manager
2. Environment variable DB_ENCRYPTION_KEY
 - Set with: Sys.setenv(DB_ENCRYPTION_KEY = "...")
 - Best for development
3. Error if neither available
 - Helpful error message with setup instructions

Value

Character: 64-char hex encryption key

Examples

```
## Not run:
# Development (environment variable):
Sys.setenv(DB_ENCRYPTION_KEY = "a1b2c3d4...")
key <- get_encryption_key()

# Production (AWS KMS):
key <- get_encryption_key(aws_kms_key_id = "arn:aws:kms:...")

## End(Not run)
```

get_encryption_key_from_aws_kms

Retrieve encryption key from AWS Secrets Manager

Description

Retrieves stored encryption key from AWS Secrets Manager. Requires paws package and AWS credentials.

Usage

```
get_encryption_key_from_aws_kms(key_id = NULL)
```

Arguments

key_id Character: AWS Secrets Manager secret name (optional)

Details

Default secret name: "zzedc/db-encryption-key"

Requirements:

- paws package installed: `install.packages("paws")`
- AWS credentials configured: `~/.aws/credentials` or environment variables
- AWS IAM permissions: `secretsmanager:GetSecretValue`

Value

Character: Decrypted encryption key (64 hex chars)

Examples

```
## Not run:
# Requires AWS credentials
key <- get_encryption_key_from_aws_kms("zzedc/db-encryption-key")

## End(Not run)
```

get_instrument_field	<i>Get instrument field by name</i>
----------------------	-------------------------------------

Description

Retrieves a single field definition from an instrument.

Usage

```
get_instrument_field(  
  instrument_name,  
  field_name,  
  instruments_dir = "instruments/"  
)
```

Arguments

instrument_name	Name of instrument
field_name	Name of field to retrieve
instruments_dir	Path to instruments directory

Value

List with field properties, or NULL if not found

get_page_summary	<i>Generate page summary statistics</i>
------------------	---

Description

Calculates column-wise statistics for paginated data

Usage

```
get_page_summary(page_data, numeric_cols = NULL)
```

Arguments

page_data	data.frame with current page data
numeric_cols	Column names to compute stats for

Value

data.frame with summary statistics

`get_setup_instructions`*Get Setup Instructions*

Description

Returns helpful instructions for completing setup

Usage

```
get_setup_instructions()
```

Value

Character string with setup instructions

`handle_error`*Handle errors with logging and user notification*

Description

Wraps expression evaluation with error handling, logging, and user feedback.

Usage

```
handle_error(  
  expr,  
  error_title = "Error",  
  show_user = TRUE,  
  log_file = NULL,  
  return_value = NULL  
)
```

Arguments

<code>expr</code>	Expression to evaluate
<code>error_title</code>	Character - title for user error message
<code>show_user</code>	Logical - show error modal to user?
<code>log_file</code>	Character - file to log errors (optional)
<code>return_value</code>	Value to return if error occurs (default: NULL)

Value

Result of `expr` if successful, `return_value` if error

Examples

```
## Not run:
result <- handle_error({
  authenticate_user(username, password)
}, error_title = "Authentication Failed")

## End(Not run)
```

import_instrument	<i>Import instrument as new form</i>
-------------------	--------------------------------------

Description

Imports a pre-built instrument into the project as a new form. Creates form record in database and returns form metadata.

Usage

```
import_instrument(
  instrument_name,
  form_name = NULL,
  form_description = NULL,
  db_conn = NULL,
  instruments_dir = "instruments/"
)
```

Arguments

instrument_name	Name of instrument to import (e.g., "phq9")
form_name	New form name (defaults to instrument name if not provided)
form_description	Description of form for display
db_conn	Database connection (RSQLite::SQLiteConnection)
instruments_dir	Path to instruments directory

Value

List containing:

- success: Logical, operation successful?
- form_id: ID of newly created form
- form_name: Name of created form
- fields_imported: Number of fields added
- message: Status message
- errors: Character vector of any errors encountered

Examples

```
## Not run:
result <- import_instrument(
  instrument_name = "phq9",
  form_name = "baseline_depression",
  form_description = "PHQ-9 administered at baseline visit",
  db_conn = conn
)

## End(Not run)
```

init

Initialize ZZedc Project

Description

Create a new ZZedc project locally or on a server. Two modes available:

- Interactive: User-friendly prompts in R console (recommended for novices)
- Config: Read from YAML configuration file (recommended for DevOps/AWS)

Usage

```
init(mode = "interactive", config_file = NULL, project_dir = ".")
```

Arguments

mode	Character. Either "interactive" (default) or "config"
config_file	Character. Path to configuration YAML file (required if mode="config")
project_dir	Character. Directory where project will be created (default: current directory)

Details

Interactive Mode:

Guides users through setup with prompts in the R console:

```
zzedc::init()
# Will ask for:
# - Study name
# - Protocol ID
# - PI information
# - Admin account details
# - Security settings
```

Config File Mode:

Reads configuration from YAML file (non-interactive):

```
zzedc::init(mode = "config", config_file = "zzedc_config.yml")
# Silently creates project from config
# Useful for automation, Docker, AWS
```

Value

Invisibly returns list with setup results and project location

Examples

```
## Not run:  
# Interactive mode (novice user)  
zzedc::init()  
  
# Config file mode (DevOps)  
zzedc::init(mode = "config", config_file = "aws_config.yml")  
  
## End(Not run)
```

init_audit_log	<i>Initialize audit log</i>
----------------	-----------------------------

Description

Creates a reactive audit log storage with immutable properties.

Usage

```
init_audit_log()
```

Value

reactiveVal containing tibble of audit records

Examples

```
## Not run:  
audit_log <- init_audit_log()  
log_audit_event(audit_log, "user1", "LOGIN", "authentication", status = "success")  
  
## End(Not run)
```

init_from_config	<i>Config File Mode: Non-Interactive Setup</i>
------------------	--

Description

Config File Mode: Non-Interactive Setup

Usage

```
init_from_config(config_file, project_dir = ".")
```

Arguments

- config_file Path to YAML configuration file
- project_dir Directory where project will be created

Value

List with setup results

init_interactive	<i>Interactive Mode: Guided Setup</i>
------------------	---------------------------------------

Description

Interactive Mode: Guided Setup

Usage

```
init_interactive(project_dir = ".")
```

Arguments

- project_dir Directory where project will be created

Value

List with setup results

init_session_timeout	<i>Initialize session timeout tracking</i>
----------------------	--

Description

Creates reactive values to track user activity and session state.

Usage

```
init_session_timeout()
```

Value

List containing reactive objects for session management

is_configured	<i>Check if ZZedc is Already Configured</i>
---------------	---

Description

Detect if ZZedc is being launched for the first time by checking for required configuration files. Returns TRUE if the system is fully configured.

Usage

```
is_configured(db_path = "../data/zzedc.db", config_path = "../config.yml")
```

Arguments

db_path	Path to database file
config_path	Path to config file

Value

Logical. TRUE if fully configured, FALSE otherwise

launch_setup_if_needed	<i>Launch Setup Mode if Needed</i>
------------------------	------------------------------------

Description

Checks if ZZedc is configured. If not, shows setup options. Called from app startup to intercept first-time users.

Usage

```
launch_setup_if_needed(  
  db_path = "../data/zzedc.db",  
  config_path = "../config.yml"  
)
```

Arguments

db_path	Path to database file
config_path	Path to config file

Value

Invisibly returns setup status

launch_zzedc	<i>Launch the ZZedc Shiny Application</i>
--------------	---

Description

This function launches the interactive 'Shiny' application for electronic data capture (EDC) in clinical trials.

Usage

```
launch_zzedc(..., launch.browser = TRUE, host = "127.0.0.1", port = NULL)
```

Arguments

...	Additional arguments passed to runApp
launch.browser	Logical, whether to launch the app in browser. Default is TRUE.
host	Character string of IP address to listen on. Default is "127.0.0.1".
port	Integer specifying the port to listen on. Default is NULL (random port).

Details

The application provides comprehensive electronic data capture for clinical trials with the following features:

- Secure user authentication with role-based access
- Data entry forms with validation and quality control
- Comprehensive reporting system (basic, quality, statistical)
- Advanced data exploration and visualization tools
- Flexible export capabilities with multiple formats
- Modern responsive design using Bootstrap 5 via bslib

Value

No return value, launches the Shiny application

Examples

```
## Not run:
# Launch the application
launch_zzedc()

# Launch on specific port
launch_zzedc(port = 3838)

# Launch without opening browser
launch_zzedc(launch.browser = FALSE)

## End(Not run)
```

```
list_available_instruments
```

List available instruments

Description

Returns names and metadata of all available pre-built instruments.

Usage

```
list_available_instruments(instruments_dir = "instruments/")
```

Arguments

```
instruments_dir
```

Path to instruments directory (default: "instruments/")

Value

data.frame with columns:

- name: Instrument ID (e.g., "phq9")
- full_name: Full instrument name
- items: Number of items
- description: Brief description

Examples

```
## Not run:
available <- list_available_instruments()
print(available)

## End(Not run)
```

```
load_instrument_template
```

Load instrument template from CSV

Description

Loads a pre-built instrument template and returns as data.frame with validated structure.

Usage

```
load_instrument_template(instrument_name, instruments_dir = "instruments/")
```

Arguments

```
instrument_name
```

Name of instrument (e.g., "phq9", "gad7")

```
instruments_dir
```

Path to instruments directory

Value

data.frame with columns:

- field_name: Unique field identifier
- field_label: User-facing label
- field_type: Input type (text, numeric, select, etc.)
- validation_rules: JSON string with validation constraints
- description: Item description/instruction text
- required: Logical, is field required?

Examples

```
## Not run:
phq9_fields <- load_instrument_template("phq9")
head(phq9_fields)

## End(Not run)
```

log_audit_event	<i>Log an audit event</i>
-----------------	---------------------------

Description

Records an audit event with cryptographic chaining to previous record.

Usage

```
log_audit_event(
  audit_log,
  user_id,
  action,
  resource,
  old_value = "",
  new_value = "",
  status = "success",
  error_message = ""
)
```

Arguments

audit_log	reactiveVal object (from init_audit_log)
user_id	Character - user performing action
action	Character - action type (e.g., "LOGIN", "DATA_EXPORT", "FORM_SUBMISSION")
resource	Character - what was affected (e.g., "authentication", "subject_123", "report_export")
old_value	Character - previous value (for modifications)
new_value	Character - new value (for modifications)
status	Character - success/failure
error_message	Character - error details if status == "failure"

Value

Invisibly returns the new record (including hash)

Examples

```
## Not run:
audit_log <- init_audit_log()
log_audit_event(
  audit_log,
  user_id = "john.doe",
  action = "LOGIN_ATTEMPT",
  resource = "authentication",
  status = "success"
)

## End(Not run)
```

log_export_event	<i>Log export event to audit trail</i>
------------------	--

Description

Records data export with details for compliance audit

Usage

```
log_export_event(user_id, data_source, format, rows, audit_log)
```

Arguments

user_id	User performing export
data_source	Source of exported data
format	Export format
rows	Number of rows exported
audit_log	Audit log reactiveVal

notify_if_invalid	<i>Validate and notify</i>
-------------------	----------------------------

Description

Checks a condition and shows notification if false.

Usage

```
notify_if_invalid(condition, message, type = "warning")
```

Arguments

condition	Logical - condition to check
message	Character - message to show if condition is FALSE
type	Character - notification type ("message", "warning", "error")

Value

Invisibly returns the condition

paginate_data	<i>Create paginated data view</i>
---------------	-----------------------------------

Description

Prepares data for paginated display with server-side processing. Handles filtering, sorting, and pagination efficiently.

Usage

```
paginate_data(
  data,
  page_size = 25,
  search_term = NULL,
  sort_by = NULL,
  sort_direction = "asc",
  page_number = 1
)
```

Arguments

data	data.frame to paginate
page_size	Number of rows per page (default: 25)
search_term	Optional text to filter rows
sort_by	Column name to sort by
sort_direction	"asc" or "desc"
page_number	Current page (1-indexed)

Value

List containing:

- data: data.frame with rows for current page
- pagination: list with page info (total_pages, total_rows, current_page)
- summary: summary statistics

Examples

```
## Not run:
paginated <- paginate_data(
  large_dataset,
  page_size = 25,
  page_number = 1
)
display_data(paginated$data)
show_page_numbers(paginated$pagination$total_pages)

## End(Not run)
```

`prepare_all_files_export`*Prepare all files data for export*

Description

Prepare all files data for export

Usage

```
prepare_all_files_export(options = NULL)
```

Arguments

`options` List of export options

Value

List of file data

`prepare_edc_export`*Prepare EDC data for export*

Description

Retrieves EDC data with optional metadata and filtering

Usage

```
prepare_edc_export(db_conn, options = NULL)
```

Arguments

`db_conn` Database connection

`options` List with `include_metadata`, `include_timestamps`, `date_range`, etc.

Value

data.frame with EDC export data

```
prepare_export_data
```

Prepare data for export

Description

Retrieves and formats data based on export configuration. Pure function with no Shiny dependencies.

Usage

```
prepare_export_data(data_source, format, options = NULL, db_conn = NULL)
```

Arguments

<code>data_source</code>	Character indicating data source ("edc", "all_files", "reports", "sample")
<code>format</code>	Character specifying export format: <ul style="list-style-type: none"> • "csv": Comma-separated values • "xlsx": Excel workbook • "json": JSON format • "sas": SAS transport file (.xpt) • "spss": SPSS/PSPP format (.sav) • "stata": Stata format (.dta) • "rds": R serialized object (.rds) • "pdf": PDF document (requires template) • "html": HTML document (requires template)
<code>options</code>	List of export options (metadata, timestamps, date_range, etc.)
<code>db_conn</code>	Database connection (if data_source == "edc")

Value

List containing:

- data: data.frame or list with export data
- info: metadata about export (rows, columns, size estimate)
- warnings: any issues encountered

Examples

```
## Not run:
export_result <- prepare_export_data(
  data_source = "edc",
  format = "csv",
  options = list(include_metadata = TRUE, include_timestamps = TRUE)
)

## End(Not run)
```

`prepare_reports_export`*Prepare reports data for export*

Description

Prepare reports data for export

Usage

```
prepare_reports_export(options = NULL)
```

Arguments

`options` List of export options

Value

List of report data

`prepare_sample_export` *Prepare sample data for export*

Description

Prepare sample data for export

Usage

```
prepare_sample_export(options = NULL)
```

Arguments

`options` List of export options

Value

data.frame with sample data

query_audit_log	<i>Query audit log</i>
-----------------	------------------------

Description

Filter audit log by various criteria.

Usage

```
query_audit_log(  
  audit_log,  
  user_id = NULL,  
  action = NULL,  
  resource = NULL,  
  start_date = NULL,  
  end_date = NULL  
)
```

Arguments

audit_log	reactiveVal or data.frame containing audit records
user_id	Character - filter by user (optional)
action	Character - filter by action type (optional)
resource	Character - filter by resource (optional)
start_date	Date - start of date range (optional)
end_date	Date - end of date range (optional)

Value

Filtered data.frame

renderPanel	<i>Render form panel with typed input fields</i>
-------------	--

Description

Generates appropriate input controls based on field metadata. Supports 15+ field types including text input, numeric fields, dates, times, email, selection lists, radio buttons, checkboxes, text areas, sliders, file uploads, and digital signatures for flexible data collection.

Usage

```
renderPanel(fields, field_metadata = NULL)
```

Arguments

fields	Character vector of field names OR list of field configurations
field_metadata	List containing field definitions with type, required, choices, etc.

Details

Field metadata format: `list(age = list(type = "numeric", required = TRUE, min = 0, max = 150), email = list(type = "email", required = TRUE), treatment = list(type = "select", choices = c("A", "B", "C")), visit_date = list(type = "date", required = TRUE), visit_time = list(type = "time", required = TRUE), pain_level = list(type = "slider", min = 0, max = 10, value = 5), symptoms = list(type = "checkbox_group", choices = c("Pain", "Fever", "Cough"))))`

Value

List of Shiny input controls matching field types

Examples

```
## Not run:
metadata <- list(
  age = list(type = "numeric", required = TRUE, label = "Age (years)"),
  gender = list(
    type = "select",
    choices = c("M", "F"),
    label = "Gender"
  ),
  pregnancy_date = list(
    type = "date",
    label = "Pregnancy Due Date",
    show_if = "gender == 'F'" # Branching logic
  ),
  visit_time = list(type = "time", required = TRUE, label = "Visit Time")
)
renderPanel(names(metadata), metadata)

## End(Not run)
```

safe_reactive

Safe reactive expression evaluation

Description

Wraps reactive expression with error handling to prevent app crashes from reactive errors.

Usage

```
safe_reactive(expr, on_error = NULL, on_empty = NULL)
```

Arguments

<code>expr</code>	Expression to evaluate reactively
<code>on_error</code>	Function to call on error (or value to return)
<code>on_empty</code>	Function or value for empty results

Value

Reactive expression result or error value

save_validated_form	<i>Save validated form data to database</i>
---------------------	---

Description

Saves validated and cleaned form data with audit logging

Usage

```
save_validated_form(conn, table_name, cleaned_data, user_id, audit_log = NULL)
```

Arguments

conn	Database connection
table_name	Table to insert into
cleaned_data	Validated data from validate_form()
user_id	User submitting the form
audit_log	Optional audit log to record submission

Value

List with success status and record ID

setup_form_validation	<i>Create server-side form validation observer</i>
-----------------------	--

Description

Sets up reactive validation that updates UI in real-time

Usage

```
setup_form_validation(
  session,
  form_fields,
  field_metadata,
  error_container_id = "form_errors"
)
```

Arguments

session	Shiny session object
form_fields	Character vector of form field names
field_metadata	List with field validation rules
error_container_id	ID of element to display errors

setup_pagination_observers	
	<i>Setup pagination observers</i>

Description

Creates reactive observers to handle pagination navigation

Usage

```
setup_pagination_observers(  
  session,  
  data_reactive,  
  current_page,  
  input_id = "data"  
)
```

Arguments

session	Shiny session object
data_reactive	Reactive expression returning current data
current_page	Reactive value holding current page number
input_id	Namespace ID for pagination inputs

sort_data	<i>Sort data frame</i>
-----------	------------------------

Description

Sorts data by specified column

Usage

```
sort_data(data, sort_column, direction = "asc")
```

Arguments

data	data.frame to sort
sort_column	Column name
direction	"asc" or "desc"

Value

Sorted data.frame

success_response	Create standardized success response
------------------	--------------------------------------

Description

Returns a consistent success response structure.

Usage

```
success_response(message = "Success", data = NULL)
```

Arguments

message	Character - success message
data	List - data to return

Value

List with success=TRUE and message

test_encryption	Test database encryption
-----------------	--------------------------

Description

Creates a test encrypted database, writes data, reads it back, and verifies encryption is actually being applied (file is binary, not plaintext).

Usage

```
test_encryption(db_path, key)
```

Arguments

db_path	Character: Path to test database file (will be created and deleted)
key	Character: Encryption key to test

Details

Test procedure:

1. Connect to database with key
2. Write test data
3. Disconnect
4. Verify file is encrypted (random bytes, not readable text)
5. Reconnect with correct key -> data readable
6. Verify data integrity
7. Cleanup temporary database

This function verifies that SQLCipher is properly compiled into RSQLite.

Value

Logical TRUE if all tests pass, otherwise stops with error

Examples

```
## Not run:
key <- generate_db_key()
test_encryption(tempfile(fileext = ".db"), key)

## End(Not run)
```

update_session_activity

Manual session activity update

Description

Explicitly update last activity time. Useful when activity isn't captured through normal input changes.

Usage

```
update_session_activity(session_tracker)
```

Arguments

session_tracker
Object returned from init_session_timeout()

validate_field_value *Validate individual field value*

Description

Type-specific validation for a single form field

Usage

```
validate_field_value(field_name, value, type, rules)
```

Arguments

field_name	Character name of field
value	Value to validate
type	Field type (text, numeric, date, email, select, checkbox)
rules	List of validation rules

Value

List with valid, message, cleaned_value, warning

validate_filename	<i>Validate and Sanitize Filename</i>
-------------------	---------------------------------------

Description

Sanitizes a filename by removing or replacing problematic characters, preventing path traversal attacks, and limiting length

Usage

```
validate_filename(filename, max_length = 100)
```

Arguments

filename	Character string to sanitize
max_length	Maximum filename length (default 100)

Value

Sanitized filename safe for use in file operations

validate_form	<i>Validate entire form submission</i>
---------------	--

Description

Validates all form fields against metadata rules. Returns detailed validation results suitable for user feedback.

Usage

```
validate_form(form_data, field_metadata)
```

Arguments

form_data	List or data.frame with submitted form values
field_metadata	List defining validation rules for each field

Value

List with:

- valid: logical, TRUE if all validations passed
- errors: named list of field-specific error messages
- warnings: named list of field-specific warnings
- cleaned_data: validated and cleaned data

Examples

```
## Not run:
metadata <- list(
  age = list(type = "numeric", required = TRUE, min = 18, max = 120),
  email = list(type = "email", required = TRUE)
)

result <- validate_form(
  list(age = 25, email = "user@example.com"),
  metadata
)

if (result$valid) {
  # Process the cleaned data
  save_record(result$cleaned_data)
} else {
  # Show errors to user
  show_validation_errors(result$errors)
}

## End(Not run)
```

 validate_instrument_csv

Validate instrument CSV structure

Description

Checks that a CSV file has correct structure for import. Used for validation before accepting user uploads.

Usage

```
validate_instrument_csv(filepath)
```

Arguments

filepath Path to CSV file to validate

Value

List containing:

- valid: Logical, structure is valid?
- errors: Character vector of validation errors
- warnings: Character vector of warnings
- field_count: Number of fields in file

verify_audit_log_integrity	<i>Verify audit log integrity</i>
----------------------------	-----------------------------------

Description

Validates that audit log hasn't been tampered with by checking hash chain. Returns TRUE if all hashes are correctly chained, FALSE otherwise.

Usage

```
verify_audit_log_integrity(audit_log)
```

Arguments

audit_log	reactiveVal or data.frame containing audit records
-----------	--

Value

Logical - TRUE if log integrity verified, FALSE if tampering detected

verify_db_key	<i>Verify database encryption key format</i>
---------------	--

Description

Validates that a key is properly formatted for SQLCipher. Checks length, characters, and format.

Usage

```
verify_db_key(key)
```

Arguments

key	Character string: The key to validate
-----	---------------------------------------

Details

Valid format requirements:

- Exactly 64 hexadecimal characters (256 bits)
- All lowercase a-f and 0-9
- Single string (length 1)

Invalid keys will stop execution with descriptive error message.

Value

Logical TRUE if valid, otherwise stops with error message

Examples

```
## Not run:
key <- generate_db_key()
verify_db_key(key) # Returns TRUE

## End(Not run)
```

%%

Null Coalesce Operator

Description

Null Coalesce Operator

Usage

`x %% y`

Arguments

<code>x</code>	First value
<code>y</code>	Default value if <code>x</code> is NULL

Value

`x` if not NULL, otherwise `y`

Index

* internal

- [%, 39](#)
- [detect_setup_status, 8](#)
- [enable_session_timeout, 9](#)
- [execute_init_setup, 10](#)
- [generate_security_salt, 13](#)
- [get_encryption_key_from_aws_kms, 14](#)
- [get_instrument_field, 15](#)
- [get_setup_instructions, 16](#)
- [init_from_config, 19](#)
- [init_interactive, 20](#)
- [init_session_timeout, 20](#)
- [is_configured, 21](#)
- [launch_setup_if_needed, 21](#)
- [update_session_activity, 35](#)
- [validate_filename, 36](#)
- [zzedc-package, 3](#)
- [%, 39](#)
- [complete_wizard_setup, 4](#)
- [create_error_display, 5](#)
- [create_launch_script, 5](#)
- [create_paginated_reactive, 6](#)
- [create_pagination_ui, 6](#)
- [create_wizard_config, 7](#)
- [create_wizard_database, 7](#)
- [create_wizard_directories, 8](#)
- [detect_setup_status, 8](#)
- [enable_session_timeout, 9](#)
- [error_response, 9](#)
- [execute_init_setup, 10](#)
- [export_audit_log, 10](#)
- [export_to_file, 11](#)
- [filter_data_by_search, 11](#)
- [generate_db_key, 12](#)
- [generate_export_filename, 12](#)
- [generate_security_salt, 13](#)
- [get_encryption_key, 13](#)
- [get_encryption_key_from_aws_kms, 14](#)
- [get_instrument_field, 15](#)

- [get_page_summary, 15](#)
- [get_setup_instructions, 16](#)
- [handle_error, 16](#)
- [import_instrument, 17](#)
- [init, 18](#)
- [init_audit_log, 19](#)
- [init_from_config, 19](#)
- [init_interactive, 20](#)
- [init_session_timeout, 20](#)
- [is_configured, 21](#)
- [launch_setup_if_needed, 21](#)
- [launch_zzedc, 22](#)
- [list_available_instruments, 23](#)
- [load_instrument_template, 23](#)
- [log_audit_event, 24](#)
- [log_export_event, 25](#)
- [notify_if_invalid, 25](#)
- [paginate_data, 26](#)
- [prepare_all_files_export, 27](#)
- [prepare_edc_export, 27](#)
- [prepare_export_data, 28](#)
- [prepare_reports_export, 29](#)
- [prepare_sample_export, 29](#)
- [query_audit_log, 30](#)
- [renderPanel, 30](#)
- [runApp, 22](#)
- [safe_reactive, 31](#)
- [save_validated_form, 32](#)
- [setup_form_validation, 32](#)
- [setup_pagination_observers, 33](#)
- [sort_data, 33](#)
- [success_response, 34](#)
- [test_encryption, 34](#)
- [update_session_activity, 35](#)
- [validate_field_value, 35](#)

`validate_filename`, [36](#)
`validate_form`, [36](#)
`validate_instrument_csv`, [37](#)
`verify_audit_log_integrity`, [38](#)
`verify_db_key`, [38](#)

`zzedc (zzedc-package)`, [3](#)
`zzedc-package`, [3](#)