Package: shinypayload (via r-universe)

May 26, 2026

Type Package

Title Accept POST Data and URL Parameters in 'shiny' (Same-Port Integration)

Version 0.3.0

Author Pawan Rama Mali [aut, cre]

Maintainer Pawan Rama Mali <prm@outlook.in>

Description Handle POST requests on a custom path (e.g., /ingress) inside the same 'shiny' HTTP server using user interface functions and HTTP responses. Expose latest payload as a reactive and provide helpers for query parameters.

License MIT + file LICENSE

URL https://github.com/PawanRamaMali/shinypayload, https://pawanramamali.github.io/shinypayload/

BugReports https://github.com/PawanRamaMali/shinypayload/issues

Encoding UTF-8

Depends R (>= 4.1)

Imports shiny (>= 1.7.4), jsonlite

Suggests testthat (>= 3.0.0), covr, styler, roxygen2, DT, digest, xml2, jsonvalidate

Config/testthat/edition 3

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.3

Config/pak/sysreqs cmake make libuv1-dev zlib1g-dev

Repository https://pawanramamali.r-universe.dev

Date/Publication 2026-01-26 15:53:23 UTC

RemoteUrl https://github.com/pawanramamali/shinypayload

RemoteRef HEAD

RemoteSha 35a376bd9ef873a3f5feff426e4b998fdc58f778

Contents

params_get *Get URL query parameters in Shiny*

Description

Get URL query parameters in Shiny

Usage

```
params_get(session, keys = NULL)
```

Arguments

session	Shiny session
keys	Optional character vector of keys to pull; if NULL return all

Value

A named list containing the URL query parameters. If keys is specified, only those parameters are returned. If no parameters exist or the specified keys are not found, returns an empty list or list with NULL values respectively.

Examples

```
if (interactive()) {
  server <- function(input, output, session) {
    # Get all query parameters
    all_params <- params_get(session)

    # Get specific parameters
    user_params <- params_get(session, keys = c("user_id", "token"))

    # Use in outputs
    output$params_display <- renderText({
      params <- params_get(session)
      if (length(params) > 0) {
        paste("Parameters:", jsonlite::toJSON(params))
      } else {
        "No parameters provided"
      }
    })
  }
}
```

payload_batch *Create a batch reactive that collects payloads and processes them in groups*

Description

Create a batch reactive that collects payloads and processes them in groups

Usage

```
payload_batch(
  path = "/ingress",
  session,
  batch_size = 10,
  batch_timeout_ms = 5000,
  process_func = NULL,
  intervalMillis = 500
)
```

Arguments

path	The URL path used in payload_ui() or payload_methods() (default "/ingress")
session	The Shiny session object
batch_size	Number of payloads to collect before processing (default 10)
batch_timeout_ms	Maximum time to wait for batch completion in ms (default 5000)
process_func	Function to process the batch of payloads
intervalMillis	Polling interval in milliseconds (default 500)

Value

A reactive expression that returns processed batch results

Examples

```
if (interactive()) {
  server <- function(input, output, session) {
    # Process sensor data in batches of 5
    sensor_batch <- payload_batch("/api/sensors", session,
      batch_size = 5,
      process_func = function(payloads) {
        temperatures <- sapply(payloads, function(p) p$payload$temperature)
        list(
          count = length(temperatures),
          avg_temp = mean(temperatures, na.rm = TRUE),
          max_temp = max(temperatures, na.rm = TRUE),
          timestamp = Sys.time()
```

```
        )
      }
    )
  }
}
```

`payload_conditional` *Create a conditional reactive that updates only when conditions are met*

Description

Create a conditional reactive that updates only when conditions are met

Usage

```
payload_conditional(
  path = "/ingress",
  session,
  condition_func,
  intervalMillis = 300
)
```

Arguments

`path`	The URL path used in payload_ui() or payload_methods() (default "/ingress")
`session`	The Shiny session object
`condition_func`	Function that returns TRUE when reactive should update
`intervalMillis`	Polling interval in milliseconds (default 300)

Value

A reactive expression that updates only when condition is met

Examples

```
if (interactive()) {
  server <- function(input, output, session) {
    # Only update when temperature exceeds threshold
    high_temp_alert <- payload_conditional("/api/sensors", session,
      condition_func = function(payload) {
        !is.null(payload$payload$temperature) &&
        payload$payload$temperature > 30
      }
    )

    # Only update during business hours
    business_hours_data <- payload_conditional("/api/data", session,
```

```
      condition_func = function(payload) {
        hour <- as.numeric(format(Sys.time(), "%H"))
        hour >= 9 && hour <= 17
      }
    )
  }
}
```

---

payload_cors_config *Configure Cross-Origin Resource Sharing (CORS)*

Description

Configure Cross-Origin Resource Sharing (CORS)

Usage

```
payload_cors_config(
  enabled = FALSE,
  origins = "*",
  methods = c("GET", "POST", "PUT", "PATCH", "DELETE", "OPTIONS"),
 headers = c("Content-Type", "Authorization", "X-Requested-With", "X-Request-ID"),
  credentials = FALSE,
  max_age = 86400
)
```

Arguments

enabled	Enable or disable CORS (default FALSE)
origins	Allowed origins. Use "*" for any origin, or provide a character vector of specific domains (default "*")
methods	HTTP methods to allow (default includes common methods)
headers	Headers to allow in requests (default includes common headers)
credentials	Allow credentials in cross-origin requests (default FALSE)
max_age	Maximum time (in seconds) browsers should cache preflight results (default 86400 = 24 hours)

Value

Invisibly returns TRUE, called for side effects

Examples

```
if (interactive()) {
  # Enable CORS for all origins
  payload_cors_config(enabled = TRUE)

  # Restrict to specific origins
  payload_cors_config(
    enabled = TRUE,
    origins = c("https://myapp.com", "https://staging.myapp.com"),
    credentials = TRUE
  )

  # Disable CORS
  payload_cors_config(enabled = FALSE)
}
```

`payload_cors_status` *Get current CORS configuration*

Description

Get current CORS configuration

Usage

```
payload_cors_status()
```

Value

A list containing current CORS settings

Examples

```
if (interactive()) {
  status <- payload_cors_status()
  cat("CORS enabled:", status$enabled, "\n")
  cat("Allowed origins:", paste(status$origins, collapse = ", "), "\n")
}
```

payload_data_clear *Clear data processing configuration*

Description

Clear data processing configuration

Usage

```
payload_data_clear(clear_hooks = TRUE, clear_limits = TRUE)
```

Arguments

clear_hooks	Whether to clear transformation hooks (default TRUE)
clear_limits	Whether to clear payload size limits (default TRUE)

Value

No return value, updates global configuration

Examples

```
if (interactive()) {
  # Clear all data processing configuration
  payload_data_clear()

  # Clear only transformation hooks
  payload_data_clear(clear_limits = FALSE)
}
```

payload_data_config *Configure data processing and transformation settings*

Description

Configure data processing and transformation settings

Usage

```
payload_data_config(transformation_hooks = NULL, max_payload_size = NULL)
```

Arguments

transformation_hooks	List of functions to apply to parsed data. Each function should accept (data, content_type, req) and return transformed data
max_payload_size	Maximum payload size in bytes (optional, for validation)

Value

No return value, updates global configuration

Examples

```
if (interactive()) {
  # Add a transformation hook to convert timestamps
  timestamp_hook <- function(data, content_type, req) {
    if (is.list(data) && !is.null(data$timestamp)) {
      data$timestamp <- as.POSIXct(data$timestamp, origin = "1970-01-01")
    }
    return(data)
  }

  # Add a validation hook
  validation_hook <- function(data, content_type, req) {
    if (is.list(data) && is.null(data$user_id)) {
      stop("user_id is required")
    }
    return(data)
  }

  payload_data_config(
    transformation_hooks = list(timestamp_hook, validation_hook),
    max_payload_size = 1024 * 1024  # 1MB limit
  )
}
```

---

payload_data_status *Get current data processing configuration*

Description

Get current data processing configuration

Usage

```
payload_data_status()
```

Value

A list containing current data processing settings

Examples

```
if (interactive()) {
  config <- payload_data_status()
  cat("Transformation hooks:", length(config$transformation_hooks %||% list()), "\n")
  cat("Max payload size:", config$max_payload_size %||% "unlimited", "\n")
}
```

payload_debug_config *Configure development and debugging settings*

Description

Configure development and debugging settings

Usage

```
payload_debug_config(
  debug_mode = FALSE,
  log_level = "INFO",
  max_log_entries = 1000
)
```

Arguments

debug_mode	Enable debug mode with verbose logging (default FALSE)
log_level	Logging level: "DEBUG", "INFO", "WARN", "ERROR" (default "INFO")
max_log_entries	Maximum number of log entries to keep (default 1000)

Value

No return value, updates global configuration

Examples

```
if (interactive()) {
  # Enable debug mode for development
  payload_debug_config(debug_mode = TRUE, log_level = "DEBUG")

  # Production settings
  payload_debug_config(debug_mode = FALSE, log_level = "WARN")
}
```

payload_debug_status *Get development and debugging status*

Description

Get development and debugging status

Usage

```
payload_debug_status()
```

Value

A list containing current debug settings

Examples

```
if (interactive()) {
  status <- payload_debug_status()
  cat("Debug mode:", status$debug_mode, "\n")
  cat("Log level:", status$log_level, "\n")
}
```

`payload_endpoint_url` *Generate the absolute URL for the payload endpoint*

Description

Generate the absolute URL for the payload endpoint

Usage

```
payload_endpoint_url(session, path = "/ingress")
```

Arguments

`session`	The Shiny session object
`path`	The URL path (default "/ingress")

Value

A character string containing the complete URL (including protocol, hostname, port, and path) where POST requests should be sent to reach this endpoint.

Examples

```
if (interactive()) {
  server <- function(input, output, session) {
    url <- payload_endpoint_url(session, "/data")
    print(paste("Send POST requests to:", url))
  }
}
```

payload_health_config *Configure health check endpoint*

Description

Configure health check endpoint

Usage

```
payload_health_config(enabled = TRUE, path = "/health")
```

Arguments

enabled	Enable or disable health check endpoint (default TRUE)
path	URL path for health check (default "/health")

Value

Invisibly returns TRUE, called for side effects

Examples

```
if (interactive()) {
  # Use custom health check path
  payload_health_config(enabled = TRUE, path = "/api/health")

  # Disable health checks
  payload_health_config(enabled = FALSE)
}
```

payload_health_status *Get current health check configuration*

Description

Get current health check configuration

Usage

```
payload_health_status()
```

Value

A list containing current health check settings

Examples

```
if (interactive()) {
  status <- payload_health_status()
  cat("Health check enabled:", status$enabled, "\n")
  cat("Health check path:", status$path, "\n")
}
```

`payload_history` *Get historical payloads for a specific endpoint*

Description

Get historical payloads for a specific endpoint

Usage

```
payload_history(path = "/ingress", limit = NULL, since = NULL)
```

Arguments

`path`	The URL path used in payload_ui() or payload_methods() (default "/ingress")
`limit`	Maximum number of historical entries to return (default NULL for all)
`since`	Only return payloads received after this timestamp (POSIXct or character)

Value

A list of historical payload entries, each containing: `id` (unique identifier), `timestamp`, `payload`, and `meta`

Examples

```
if (interactive()) {
  # Get last 10 payloads
  recent_payloads <- payload_history("/api/data", limit = 10)

  # Get payloads from last hour
  since_time <- Sys.time() - 3600
  recent_payloads <- payload_history("/api/data", since = since_time)

  # Process historical data
  for (entry in recent_payloads) {
    cat("ID:", entry$id, "Time:", entry$timestamp, "\n")
    print(entry$payload)
  }
}
```

payload_history_clear *Clear payload history for specific endpoint or all endpoints*

Description

Clear payload history for specific endpoint or all endpoints

Usage

```
payload_history_clear(path = NULL)
```

Arguments

path	The URL path to clear history for, or NULL to clear all (default NULL)

Value

Number of entries that were cleared

Examples

```
if (interactive()) {
  # Clear history for specific endpoint
  cleared_count <- payload_history_clear("/api/data")

  # Clear all history
  total_cleared <- payload_history_clear()
}
```

payload_history_config *Configure payload history retention policies*

Description

Configure payload history retention policies

Usage

```
payload_history_config(max_items = 100, max_age_hours = 24)
```

Arguments

max_items	Maximum number of payload entries to keep per endpoint (default 100)
max_age_hours	Maximum age in hours for payload entries (default 24)

Value

No return value, updates global configuration

Examples

```
if (interactive()) {
  # Keep more history items but for shorter time
  payload_history_config(max_items = 500, max_age_hours = 12)

  # Long-term storage with fewer items
  payload_history_config(max_items = 50, max_age_hours = 168) # 1 week
}
```

---

payload_history_stats *Get payload history statistics*

Description

Get payload history statistics

Usage

```
payload_history_stats(path = NULL)
```

Arguments

path	The URL path to get statistics for, or NULL for all endpoints (default NULL)

Value

A list containing statistics: total_entries, oldest_timestamp, newest_timestamp, endpoints (if path is NULL), and size_estimate

Examples

```
if (interactive()) {
  # Get stats for specific endpoint
  stats <- payload_history_stats("/api/data")
  cat("Total entries:", stats$total_entries, "\n")

  # Get overall stats
  overall_stats <- payload_history_stats()
  cat("Total endpoints:", length(overall_stats$endpoints), "\n")
}
```

payload_last *Get a reactive that polls for new payload data*

Description

Get a reactive that polls for new payload data

Usage

```
payload_last(path = "/ingress", session, intervalMillis = 300,
  scope = c("global", "session"))
```

Arguments

path	The URL path used in payload_ui() (default "/ingress")
session	The Shiny session object
intervalMillis	Polling interval in milliseconds (default 300)
scope	Data scope: "global" (shared across sessions, default) or "session" (isolated per session)

Value

A reactive expression (class "reactive") that returns a list with two elements when new data is available: `payload` (the parsed request body) and `meta` (metadata including timestamp, remote address, headers, etc.), or `NULL` if no data has been received yet.

Examples

```
if (interactive()) {
  server <- function(input, output, session) {
    # Global scope - data shared across all sessions
    global_data <- payload_last("/data", session)

    # Session scope - data isolated to this session only
    session_data <- payload_last("/user-data", session, scope = "session")

    observeEvent(global_data(), {
      data <- global_data()
      if (!is.null(data)) {
        print(data$payload)
        print(data$meta$timestamp)
      }
    })
  }
}
```

payload_logs *Get recent log entries*

Description

Get recent log entries

Usage

```
payload_logs(level = NULL, limit = 50, since = NULL)
```

Arguments

level	Filter by log level (optional)
limit	Maximum number of entries to return (default 50)
since	Only return logs after this timestamp (optional)

Value

A list of log entries

Examples

```
if (interactive()) {
  # Get last 20 log entries
  recent_logs <- payload_logs(limit = 20)

  # Get only error logs
  error_logs <- payload_logs(level = "ERROR")

  # Get logs from last hour
  recent_logs <- payload_logs(since = Sys.time() - 3600)
}
```

payload_logs_clear *Clear log entries*

Description

Clear log entries

Usage

```
payload_logs_clear(level = NULL)
```

Arguments

level	Clear only logs of this level (optional, clears all if NULL)

Value

Number of log entries that were cleared

Examples

```
if (interactive()) {
  # Clear all logs
  cleared_count <- payload_logs_clear()

  # Clear only debug logs
  debug_cleared <- payload_logs_clear(level = "DEBUG")
}
```

---

payload_methods *Enhanced HTTP methods support for multiple endpoints*

Description

Enhanced HTTP methods support for multiple endpoints

Usage

```
payload_methods(base_ui, endpoints)
```

Arguments

base_ui	The original UI (tagList, fluidPage, or a function(req) returning UI)
endpoints	A list of endpoint configurations. Each element should be a list with: path, methods (character vector), and optionally token

Value

A function that takes a request object and returns either the regular UI (for GET requests) or an HTTP response (for other HTTP methods). This function should be passed to shinyApp() as the ui parameter.

Examples

```
if (interactive()) {
  endpoints <- list(
    list(path = "/api/data", methods = c("POST", "PUT"), token = "secret"),
    list(path = "/api/delete", methods = "DELETE", token = "admin-token"),
    list(path = "/webhooks", methods = c("POST", "PATCH"))
  )
```

```
  ui <- payload_methods(fluidPage(h1("My App")), endpoints)
  shinyApp(ui, server, uiPattern = ".*")
}
```

---

payload_queue *Enqueue a payload for async processing*

Description

Enqueue a payload for async processing

Usage

```
payload_queue(path, payload, priority = c("normal", "high", "low"),
  correlation_id = NULL)
```

Arguments

path	The endpoint path this payload is associated with
payload	The payload data to queue
priority	Queue priority: "high", "normal", or "low" (default "normal")
correlation_id	Optional correlation ID for tracking

Value

A list with success status and queue_id if successful

Examples

```
if (interactive()) {
  # Enqueue a payload
  result <- payload_queue("/api/process", list(data = "test"))
  if (result$success) {
    cat("Queued with ID:", result$queue_id, "\n")
  }
}
```

payload_queue_clear *Clear queue items*

Description

Clear queue items

Usage

```
payload_queue_clear(include_dead_letter = FALSE)
```

Arguments

include_dead_letter	Also clear dead letter queue (default FALSE)

Value

Number of items cleared

Examples

```
if (interactive()) {
  # Clear pending queue items
  cleared <- payload_queue_clear()

  # Clear everything including dead letter
  cleared <- payload_queue_clear(include_dead_letter = TRUE)
}
```

payload_queue_config *Configure async queue processing*

Description

Configure async queue processing

Usage

```
payload_queue_config(
  enabled = FALSE,
  max_size = 1000,
  retry_attempts = 3,
  retry_delay_ms = 1000
)
```

Arguments

enabled	Enable or disable queue processing (default FALSE)
max_size	Maximum queue size (default 1000)
retry_attempts	Number of retry attempts for failed processing (default 3)
retry_delay_ms	Delay between retries in milliseconds (default 1000)

Value

Invisibly returns TRUE, called for side effects

Examples

```
if (interactive()) {
  # Enable queue with custom settings
  payload_queue_config(
    enabled = TRUE,
    max_size = 500,
    retry_attempts = 5
  )
}
```

---

payload_queue_dead_letter

Get dead letter queue items

Description

Get dead letter queue items

Usage

```
payload_queue_dead_letter(limit = 50)
```

Arguments

limit	Maximum number of items to return (default 50)

Value

A list of failed queue items

Examples

```
if (interactive()) {
  dead_items <- payload_queue_dead_letter()
  for (item in dead_items) {
    cat("Failed item:", item$id, "Error:", item$error, "\n")
  }
}
```

payload_queue_process *Process queued payloads*

Description

Process queued payloads

Usage

```
payload_queue_process(processor, max_items = 10)
```

Arguments

processor	A function that accepts (path, payload, meta) and processes it. Should return TRUE on success, FALSE on failure.
max_items	Maximum number of items to process (default 10)

Value

A list with processing results

Examples

```
if (interactive()) {
  # Process queued items
  result <- payload_queue_process(function(path, payload, meta) {
    cat("Processing payload for", path, "\n")
    # Your processing logic here
    TRUE  # Return TRUE on success
  })
  cat("Processed:", result$processed, "items\n")
}
```

payload_queue_status *Get queue status and statistics*

Description

Get queue status and statistics

Usage

```
payload_queue_status()
```

Value

A list containing queue statistics

Examples

```
if (interactive()) {
  status <- payload_queue_status()
  cat("Pending items:", status$pending, "\n")
  cat("Dead letter items:", status$dead_letter, "\n")
}
```

`payload_response_config` *Configure custom response handler for an endpoint*

Description

Configure custom response handler for an endpoint

Usage

```
payload_response_config(path, handler)
```

Arguments

`path`	The URL path to configure response handler for
`handler`	A function that accepts (payload, req) and returns a list with: `status` (HTTP status code), body (response body), `content_type` (MIME type), `headers` (additional headers). Pass NULL to remove the handler.

Value

Invisibly returns TRUE, called for side effects

Examples

```
if (interactive()) {
  # Set custom response handler
  payload_response_config("/api/process", function(payload, req) {
    list(
      status = 201L,
      body = list(
        received = TRUE,
        id = paste0("item_", sample(10000:99999, 1)),
        timestamp = Sys.time()
      )
    )
  })

  # Remove handler (use default response)
  payload_response_config("/api/process", NULL)
}
```

payload_response_status *Get current response handlers*

Description

Get current response handlers

Usage

```
payload_response_status()
```

Value

A list of configured response handlers by path

Examples

```
if (interactive()) {
  status <- payload_response_status()
  cat("Configured handlers for:", paste(names(status$handlers), collapse = ", "), "\n")
}
```

payload_schema_config *Configure JSON Schema validation for an endpoint*

Description

Configure JSON Schema validation for an endpoint

Usage

```
payload_schema_config(path, schema)
```

Arguments

path	The URL path to configure schema validation for
schema	JSON Schema as a character string, list, or path to a schema file. Pass NULL to remove schema validation.

Value

Invisibly returns TRUE, called for side effects

Examples

```
if (interactive()) {
  # Set JSON Schema for validation
  payload_schema_config("/api/data", '{
    "type": "object",
    "required": ["name", "value"],
    "properties": {
      "name": {"type": "string"},
      "value": {"type": "number"}
    }
  }')

  # Remove schema validation
  payload_schema_config("/api/data", NULL)
}
```

`payload_schema_status` *Get current JSON Schema configurations*

Description

Get current JSON Schema configurations

Usage

```
payload_schema_status()
```

Value

A list of configured schemas by path

Examples

```
if (interactive()) {
  status <- payload_schema_status()
  cat("Schemas configured for:", paste(status$configured_paths, collapse = ", "), "\n")
}
```

payload_security_clear_rate_limits

Clear rate limit records for specific IP or all IPs

Description

Clear rate limit records for specific IP or all IPs

Usage

```
payload_security_clear_rate_limits(ip_address = NULL)
```

Arguments

ip_address	Specific IP address to clear, or NULL for all (default NULL)

Value

Number of IP records that were cleared

Examples

```
if (interactive()) {
  # Clear rate limits for specific IP
  cleared <- payload_security_clear_rate_limits("192.168.1.10")

  # Clear all rate limit records
  total_cleared <- payload_security_clear_rate_limits()
}
```

payload_security_config

Configure security settings for payload endpoints

Description

Configure security settings for payload endpoints

Usage

```
payload_security_config(
  hmac_secret = NULL,
  ip_whitelist = NULL,
  ip_blacklist = NULL,
  rate_limit_enabled = FALSE,
  rate_limit_requests = 100,
  rate_limit_window_seconds = 3600
)
```

Arguments

hmac_secret	Secret key for HMAC signature validation (optional)
ip_whitelist	Character vector of allowed IP addresses (optional)
ip_blacklist	Character vector of denied IP addresses (optional)
rate_limit_enabled	Enable rate limiting (default FALSE)
rate_limit_requests	Maximum requests per window (default 100)
rate_limit_window_seconds	Time window in seconds (default 3600 = 1 hour)

Value

No return value, updates global security configuration

Examples

```
if (interactive()) {
  # Enable HMAC signature validation
  payload_security_config(hmac_secret = "your-webhook-secret")

  # IP whitelist for production
  payload_security_config(ip_whitelist = c("192.168.1.10", "10.0.0.5"))

  # Rate limiting
  payload_security_config(
    rate_limit_enabled = TRUE,
    rate_limit_requests = 50,
    rate_limit_window_seconds = 1800
  )
}
```

---

payload_security_status

Get current security configuration

Description

Get current security configuration

Usage

```
payload_security_status()
```

Value

A list containing current security settings

Examples

```
if (interactive()) {
  config <- payload_security_status()
  cat("Rate limiting enabled:", config$rate_limit_enabled, "\n")
  cat("IP whitelist count:", length(config$ip_whitelist %||% character(0)), "\n")
}
```

---

payload_stream *Create a streaming reactive for real-time payload updates*

Description

Create a streaming reactive for real-time payload updates

Usage

```
payload_stream(
  path = "/ingress",
  session,
  filter_func = NULL,
  transform_func = NULL,
  intervalMillis = 100,
  max_items = 50
)
```

Arguments

path	The URL path used in payload_ui() or payload_methods() (default "/ingress")
session	The Shiny session object
filter_func	Optional function to filter payloads. Should return TRUE to include payload
transform_func	Optional function to transform payloads before returning
intervalMillis	Polling interval in milliseconds (default 100 for real-time)
max_items	Maximum number of items to keep in stream (default 50)

Value

A reactive expression that returns a list of recent payloads matching the filter

Examples

```
if (interactive()) {
  server <- function(input, output, session) {
    # Stream all payloads
    all_stream <- payload_stream("/api/data", session)

    # Stream only error events
```

```
    error_stream <- payload_stream("/api/data", session,
      filter_func = function(payload) {
        !is.null(payload$payload$level) && payload$payload$level == "error"
      }
    )

    # Stream with transformation
    temp_stream <- payload_stream("/api/sensors", session,
      filter_func = function(payload) {
        !is.null(payload$payload$type) && payload$payload$type == "temperature"
      },
      transform_func = function(payload) {
        list(
          timestamp = payload$meta$timestamp,
          temp_celsius = payload$payload$value,
          temp_fahrenheit = payload$payload$value * 9/5 + 32
        )
      }
    )
  }
}
```

---

payload_system_status *Get comprehensive system status and diagnostics*

Description

Get comprehensive system status and diagnostics

Usage

```
payload_system_status()
```

Value

A list containing detailed system information

Examples

```
if (interactive()) {
  status <- payload_system_status()
  print(status)
}
```

payload_ui *Wrap an existing UI with an integrated POST handler on the same port*

Description

Wrap an existing UI with an integrated POST handler on the same port

Usage

```
payload_ui(base_ui, path = "/ingress", token = NULL, response_handler = NULL)
```

Arguments

base_ui	The original UI (tagList, fluidPage, or a function(req) returning UI)
path	The URL path to handle POST requests (default "/ingress")
token	Optional authentication token for POST requests
response_handler	Optional function to customize responses. Should accept (payload, req) and return a list with: status, body, content_type, headers (all optional)

Value

A function that takes a request object and returns either the regular UI (for GET requests) or an HTTP response (for POST requests). This function should be passed to shinyApp() as the ui parameter.

Examples

```
if (interactive()) {
  ui <- payload_ui(
    fluidPage(h1("My App")),
    path = "/data",
    token = "secret123"
  )
  shinyApp(ui, server, uiPattern = ".*")

  # With custom response handler
  ui <- payload_ui(
    fluidPage(h1("My App")),
    path = "/api/process",
    response_handler = function(payload, req) {
      list(
        status = 201L,
        body = list(received = TRUE, id = sample(10000:99999, 1))
      )
    }
  )
}
```

payload_upload_config *Configure multipart/file upload settings*

Description

Configure multipart/file upload settings

Usage

```
payload_upload_config(max_size = 10 * 1024 * 1024)
```

Arguments

max_size	Maximum upload size in bytes (default 10MB)

Value

Invisibly returns TRUE, called for side effects

Examples

```
if (interactive()) {
  # Allow uploads up to 50MB
  payload_upload_config(max_size = 50 * 1024 * 1024)
}
```

payload_upload_status *Get current upload configuration*

Description

Get current upload configuration

Usage

```
payload_upload_status()
```

Value

A list containing upload settings

Examples

```
if (interactive()) {
  status <- payload_upload_status()
  cat("Max upload size:", status$max_size_mb, "MB\n")
}
```

`setup_payload_endpoint` *Setup POST endpoint in server function - MUST be called in server*

Description

Setup POST endpoint in server function - MUST be called in server

Usage

```
setup_payload_endpoint(path = "/ingress", session, token = NULL)
```

Arguments

`path`	The URL path to handle POST requests (default "/ingress")
`session`	The Shiny session object
`token`	Optional authentication token for POST requests

Value

No return value, called for side effects. Registers a POST endpoint handler with the Shiny session that will process incoming requests.

Index