@push.rocks/smartlog

A minimalistic, distributed, and extensible logging tool supporting centralized log management.

- readme.md for @push.rocks/smartlog
- changelog.md for @push.rocks/smartlog

readme.md for @push.rocks/smartlog

The ultimate TypeScript logging solution for modern applications

npm version License: MIT

> **smartlog** is a powerful, distributed, and extensible logging system designed for the cloud-native era. Whether you're debugging locally, monitoring production systems, or building complex microservices, smartlog adapts to your needs with style. □□

Issue Reporting and Security

For reporting bugs, issues, or security vulnerabilities, please visit community.foss.global/. This is the central community hub for all issue reporting. Developers who sign and comply with our contribution agreement and go through identification can also get a code.foss.global/ account to submit Pull Requests directly.

□□ Why smartlog?

- □□ **Beautiful Console Output**: Color-coded, formatted logs that are actually readable
- □□ **Extensible Architecture**: Plug in any destination — databases, files, remote servers
- □□ **Distributed by Design**: Built for microservices with correlation and context tracking
- ⚡ **Zero-Config Start**: Works out of the box, scales when you need it
- □□ **Interactive CLI Tools**: Spinners and progress bars that handle non-TTY environments gracefully
- □□ **Structured Logging**: JSON-based for easy parsing and analysis
- □□ **Smart Filtering**: Log levels and context-based filtering

□□ Installation

```
# Using pnpm (recommended)
pnpm add @push.rocks/smartlog

# Using npm
npm install @push.rocks/smartlog
```

□□ Quick Start

Your First Logger

```
import { Smartlog } from '@push.rocks/smartlog';

// Create a logger with context
const logger = new Smartlog({
  logContext: {
    company: 'MyStartup',
    companyunit: 'Backend Team',
    containerName: 'api-gateway',
    environment: 'production',
    runtime: 'node',
    zone: 'eu-central'
  }
});

// Enable console output
logger.enableConsole();

// Start logging!
await logger.log('info', '□□Application started successfully');
await logger.log('error', '□□Database connection failed', {
  errorCode: 'DB_TIMEOUT',
  attemptCount: 3
});
```

Using `createForCommitinfo`

If you're integrating with a build system that provides commit info, you can use the static factory method:

```
import { Smartlog } from '@push.rocks/smartlog';

const logger = Smartlog.createForCommitinfo({
  name: 'my-app',
  version: '1.0.0',
  description: 'My application'
});

logger.enableConsole();
await logger.log('lifecycle', '□□App starting...');
```

□□ Core Concepts

Log Levels

smartlog supports semantic log levels for different scenarios:

```
// Lifecycle events
await logger.log('lifecycle', '□□Container starting up...');

// Success states
await logger.log('success', '□ Payment processed');
await logger.log('ok', '□□Health check passed');

// Information and debugging
await logger.log('info', '□□User profile updated');
await logger.log('note', '□□Cache invalidated');
await logger.log('debug', '□□Query execution plan', { sql: 'SELECT * FROM users' });

// Warnings and errors
await logger.log('warn', '⚠□  Memory usage above 80%');
await logger.log('error', '□ Failed to send email');
```

```
// Verbose output
await logger.log('silly', '□□Entering function processPayment()');
```

Available levels (from most to least verbose): `silly`, `debug`, `info`, `note`, `ok`, `success`, `warn`, `error`, `lifecycle`

Log Types

Each log entry has a type that describes what kind of data it represents:

Type	Description
`log`	Standard log message
`increment`	Counter/metric increment
`gauge`	Gauge measurement
`error`	Error event
`success`	Success event
`value`	Value recording
`finance`	Financial transaction
`compliance`	Compliance event

Log Groups for Correlation

Perfect for tracking request flows through your system:

```
// Track a user request through multiple operations
const requestGroup = logger.createLogGroup('req-7f3a2b');

requestGroup.log('info', 'Received POST /api/users');
requestGroup.log('debug', 'Validating request body');
requestGroup.log('info', 'Creating user in database');
requestGroup.log('success', 'User created successfully', { userId: 'usr_123' });

// All logs in the group share the same group ID and transaction ID
```

Log Context

Every log carries contextual information about the environment it was created in:

```
interface ILogContext {
  commitinfo?: ICommitInfo;  // Build/version info
  company?: string;           // Company name
  companyunit?: string;       // Team or department
  containerName?: string;     // Container/service name
  environment?: 'local' | 'test' | 'staging' | 'production';
  runtime?: 'node' | 'chrome' | 'rust' | 'deno' | 'cloudflare_workers';
  zone?: string;              // Deployment zone/region
}
```

□□ Log Destinations

smartlog routes log packages to any number of destinations simultaneously. Each destination implements the `ILogDestination` interface with a single `handleLog` method.

Built-in Destinations

□□ Local Console (Enhanced)

Beautiful, color-coded output for local development:

```
import { DestinationLocal } from '@push.rocks/smartlog/destination-local';

const localDestination = new DestinationLocal();
logger.addLogDestination(localDestination);

// Output is color-coded per log level:
// □□info: blue prefix, white text
// □□error: red prefix, red text
// □□ok/success: green prefix, green text
// □□warn: orange prefix, orange text
// □□note/debug: pink prefix, pink text
// □□silly: blue background, blue text
```

The `DestinationLocal` also supports **reduced logging** — a mode where repeated identical log messages are suppressed:

```
const localDest = new DestinationLocal();

localDest.logReduced('Waiting for connection...'); // Logged
localDest.logReduced('Waiting for connection...'); // Suppressed
localDest.logReduced('Waiting for connection...'); // Suppressed
localDest.logReduced('Connected!');                // Logged (new message)
```

File Logging

Persist logs to files with timestamped entries:

```
import { SmartlogDestinationFile } from '@push.rocks/smartlog/destination-file';

// Path MUST be absolute
const fileDestination = new SmartlogDestinationFile('/var/log/myapp/app.log');
logger.addLogDestination(fileDestination);

// Log entries are written as timestamped lines:
// 2024-01-15T10:30:00.000Z: Application started
// 2024-01-15T10:30:01.123Z: Processing request
```

Browser DevTools

Optimized for browser environments with styled console output:

```
import { SmartlogDestinationDevtools } from '@push.rocks/smartlog/destination-devtools';

const devtools = new SmartlogDestinationDevtools();
logger.addLogDestination(devtools);

// Uses CSS styling in browser console for beautiful, categorized output
// Different colors for error (red), info (pink), ok (green), success (green),
// warn (orange), and note (blue) levels
```

ClickHouse Analytics

Store logs in ClickHouse for powerful time-series analytics:

```
import { SmartlogDestinationClickhouse } from '@push.rocks/smartlog/destination-clickhouse';

const clickhouse = await SmartlogDestinationClickhouse.createAndStart({
```

```
  url: 'https://analytics.example.com:8123',
  database: 'logs',
  username: 'logger',
  password: process.env.CLICKHOUSE_PASSWORD
});

logger.addLogDestination(clickhouse);
```

□□ Remote Receiver

Send logs to a centralized logging service with authenticated transport:

```
import { SmartlogDestinationReceiver } from '@push.rocks/smartlog/destination-receiver';

const receiver = new SmartlogDestinationReceiver({
  passphrase: process.env.LOG_PASSPHRASE,
  receiverEndpoint: 'https://logs.mycompany.com/ingest'
});

logger.addLogDestination(receiver);
```

Logs are sent as authenticated JSON payloads with SHA-256 hashed passphrases.

□□ Custom Destinations

Build your own destination for any logging backend by implementing the `ILogDestination` interface:

```
import type { ILogDestination, ILogPackage } from '@push.rocks/smartlog/interfaces';

class ElasticsearchDestination implements ILogDestination {
  async handleLog(logPackage: ILogPackage): Promise<void> {
    await this.client.index({
      index: `logs-${new Date().toISOString().split('T')[0]}`,
      body: {
        '@timestamp': logPackage.timestamp,
        level: logPackage.level,
        message: logPackage.message,
        context: logPackage.context,
        data: logPackage.data,
```

```
        correlation: logPackage.correlation
      }
    });
  }
}

logger.addLogDestination(new ElasticsearchDestination());
```

□□ Interactive Console Features

Spinners

Create beautiful loading animations that degrade gracefully in CI/CD:

```
import { SmartlogSourceInteractive } from '@push.rocks/smartlog/source-interactive';

const spinner = new SmartlogSourceInteractive();

// Basic usage
spinner.text('□□Fetching data from API...');
// ... perform async operation
spinner.finishSuccess('□ Data fetched successfully!');

// Chain success and move to next task
spinner.text('□□Connecting to database');
// ... connect
spinner.successAndNext('□□Running migrations');
// ... migrate
spinner.finishSuccess('□□Database ready!');

// Customize appearance
spinner
  .setSpinnerStyle('dots')  // 'dots' | 'line' | 'star' | 'simple'
  .setColor('cyan')         // any terminal color
  .setSpeed(80);            // animation speed in ms

spinner.text('□□Deploying application...');
```

```
// Handle failures
try {
  await deployApp();
  spinner.finishSuccess('□ Deployed!');
} catch (error) {
  spinner.finishFail('□ Deployment failed');
}
```

Progress Bars

Track long-running operations with style:

```
import { SmartlogProgressBar } from '@push.rocks/smartlog/source-interactive';

// Create a progress bar
const progress = new SmartlogProgressBar({
  total: 100,
  width: 40,          // Bar width in characters
  complete: '█',      // Fill character
  incomplete: '░',    // Empty character
  showEta: true,      // Show estimated time remaining
  showPercent: true,  // Show percentage
  showCount: true     // Show current/total count
});

// Update progress
for (let i = 0; i <= 100; i++) {
  progress.update(i);
  await someAsyncWork();
}

progress.complete();

// Or use increment for simpler tracking
const files = await getFiles();
const fileProgress = new SmartlogProgressBar({ total: files.length });

for (const file of files) {
```

```
  await processFile(file);
  fileProgress.increment();
}

fileProgress.complete();
```

Non-Interactive Fallback

Both spinners and progress bars automatically detect non-interactive environments (CI/CD, Docker logs, piped output) and fall back to simple text output:

```
[Loading] Connecting to database
[Success] Connected to database
[Loading] Running migrations
Progress: 25% (25/100)
Progress: 50% (50/100)
Progress: 100% (100/100)
Completed: 100% (100/100)
```

Detection checks for: TTY capability, CI environment variables (GitHub Actions, Jenkins, GitLab CI, Travis, CircleCI), and `TERM=dumb`.

Backward Compatibility

The `SmartlogSourceOra` class extends `SmartlogSourceInteractive` and provides a compatibility layer for code that previously used the `ora` npm package:

```
import { SmartlogSourceOra } from '@push.rocks/smartlog/source-interactive';

const ora = new SmartlogSourceOra();
ora.oraInstance.start();
ora.oraInstance.succeed('Done!');
```

□□ Advanced Features

Minimum Log Level

Set a minimum log level that destinations can use to filter messages:

```
const logger = new Smartlog({
  logContext: { environment: 'production', runtime: 'node' },
  minimumLogLevel: 'warn'  // Destinations can check this to filter
});

// The minimumLogLevel is available as a public property
console.log(logger.minimumLogLevel); // 'warn'
```

Increment Logging

Track metrics and counters alongside your logs:

```
// Track API calls
logger.increment('info', 'api.requests', { endpoint: '/users', method: 'GET' });

// Track error types
logger.increment('error', 'payment.failed', { reason: 'insufficient_funds' });
```

Increment logs are routed to all destinations with `type: 'increment'` so analytics backends can aggregate them.

Capture All Console Output

Redirect all `console.log` and `console.error` through smartlog:

```
logger.enableConsole({
  captureAll: true
});

console.log('This goes through smartlog as info!');
console.error('This goes through smartlog as error!');

// Strings containing "Error:" are automatically classified as error level
// All captured output is prefixed with "LOG =>" to prevent recursion
```

Log Receiver Server

Accept authenticated log packages from remote services:

```
import { SmartlogReceiver } from '@push.rocks/smartlog/receiver';

const receiver = new SmartlogReceiver({
  smartlogInstance: logger,
  passphrase: 'shared-secret',
  validatorFunction: async (logPackage) => {
    // Custom validation logic
    return logPackage.context.company === 'MyCompany';
  }
});

// Handle incoming authenticated log packages (e.g., from an HTTP endpoint)
app.post('/logs', async (req, res) => {
  const result = await receiver.handleAuthenticatedLog(req.body);
  res.json(result); // { status: 'ok' } or { status: 'error' }
});

// Or handle batches
app.post('/logs/batch', async (req, res) => {
  await receiver.handleManyAuthenticatedLogs(req.body);
  res.json({ status: 'ok' });
});
```

□□ Real-World Examples

Microservice with Distributed Tracing

```
import { Smartlog } from '@push.rocks/smartlog';
import { SmartlogDestinationClickhouse } from '@push.rocks/smartlog/destination-clickhouse';
import { DestinationLocal } from '@push.rocks/smartlog/destination-local';

// Initialize logger with service context
const logger = new Smartlog({
  logContext: {
    company: 'TechCorp',
```

```
    companyunit: 'Platform',
    containerName: 'user-service',
    environment: 'production',
    runtime: 'node',
    zone: 'eu-central'
  }
});

// Add ClickHouse for analytics
const clickhouse = await SmartlogDestinationClickhouse.createAndStart({
  url: process.env.CLICKHOUSE_URL,
  database: 'microservices_logs'
});
logger.addLogDestination(clickhouse);

// Add local console for container stdout
logger.addLogDestination(new DestinationLocal());

// Express middleware for request tracking
app.use((req, res, next) => {
  const logGroup = logger.createLogGroup(req.headers['x-request-id'] || 'unknown');

  logGroup.log('info', 'Incoming request', {
    method: req.method,
    path: req.path,
    ip: req.ip
  });

  res.on('finish', () => {
    logGroup.log('info', 'Request completed', {
      statusCode: res.statusCode,
      duration: Date.now() - req.startTime
    });
  });

  next();
});
```

CLI Tool with Progress Tracking

```
import { Smartlog } from '@push.rocks/smartlog';
import { SmartlogSourceInteractive, SmartlogProgressBar } from '@push.rocks/smartlog/source-interactive';

const logger = new Smartlog({
  logContext: {
    containerName: 'migration-tool',
    environment: 'local',
    runtime: 'node'
  }
});

logger.enableConsole();

async function migrateDatabase() {
  const spinner = new SmartlogSourceInteractive();

  spinner.text('□□Connecting to database...');
  await connectDB();
  spinner.finishSuccess('□ Connected to database');

  spinner.text('□□Loading migrations...');
  const migrations = await getMigrations();
  spinner.finishSuccess(`□ Found ${migrations.length} migrations`);

  const progress = new SmartlogProgressBar({
    total: migrations.length,
    width: 40,
    showEta: true
  });

  for (const [index, migration] of migrations.entries()) {
    await logger.log('info', `Running migration: ${migration.name}`);
    await runMigration(migration);
    progress.update(index + 1);
  }

  progress.complete();
  await logger.log('success', '□□All migrations completed successfully!');
```

```
}
```

Production Logging with Multiple Destinations

```
import { Smartlog } from '@push.rocks/smartlog';
import { DestinationLocal } from '@push.rocks/smartlog/destination-local';
import { SmartlogDestinationFile } from '@push.rocks/smartlog/destination-file';
import { SmartlogDestinationReceiver } from '@push.rocks/smartlog/destination-receiver';

const logger = new Smartlog({
  logContext: {
    company: 'Enterprise Corp',
    containerName: 'payment-processor',
    environment: 'production',
    runtime: 'node',
    zone: 'us-east-1'
  },
  minimumLogLevel: 'info'
});

// Color-coded console for container logs
logger.addLogDestination(new DestinationLocal());

// File for audit trail
logger.addLogDestination(new SmartlogDestinationFile('/var/log/app/audit.log'));

// Central logging service
logger.addLogDestination(new SmartlogDestinationReceiver({
  passphrase: process.env.LOG_PASSPHRASE,
  receiverEndpoint: 'https://logs.enterprise.com/ingest'
}));

// Custom inline destination for critical alerts
logger.addLogDestination({
  async handleLog(logPackage) {
    if (logPackage.level === 'error' && logPackage.data?.critical) {
```

```
      await sendSlackAlert(`□□Critical error: ${logPackage.message}`);
    }
  }
});
```

□□ API Reference

Smartlog Class

```
class Smartlog {
  // Static factory
  static createForCommitinfo(commitinfo: ICommitInfo): Smartlog;

  // Constructor
  constructor(options: {
    logContext: ILogContext;
    minimumLogLevel?: TLogLevel;  // default: 'silly'
  });

  // Logging
  log(level: TLogLevel, message: string, data?: any, correlation?: ILogCorrelation):
Promise<void>;
  increment(level: TLogLevel, message: string, data?: any, correlation?: ILogCorrelation):
void;

  // Configuration
  enableConsole(options?: { captureAll: boolean }): void;
  addLogDestination(destination: ILogDestination): void;

  // Correlation
  createLogGroup(transactionId?: string): LogGroup;

  // Forwarding (for receiver pattern)
  handleLog(logPackage: ILogPackage): Promise<void>;

  // Instance identity
  uniInstanceId: string;
```

```
  logContext: ILogContext;
  minimumLogLevel: TLogLevel;
}
```

LogGroup Class

```
class LogGroup {
  groupId: string;          // Auto-generated unique ID
  transactionId: string;    // The transaction ID passed at creation
  smartlogRef: Smartlog;    // Reference to the parent Smartlog

  log(level: TLogLevel, message: string, data?: any): void;
}
```

ILogDestination Interface

```
interface ILogDestination {
  handleLog(logPackage: ILogPackage): Promise<void>;
}
```

ILogPackage Interface

```
interface ILogPackage<T = unknown> {
  timestamp: number;             // Unix timestamp in milliseconds
  type: TLogType;                // 'log' | 'increment' | 'gauge' | ...
  context: ILogContext;          // Environment context
  level: TLogLevel;              // Log severity
  correlation: ILogCorrelation;  // Correlation metadata
  message: string;               // The log message
  data?: T;                      // Optional structured data
}
```

Available Log Levels

Level	Description	Use Case

`silly`	Ultra-verbose	Function entry/exit, variable dumps
`debug`	Debug info	Development-time diagnostics
`info`	Informational	Standard operational messages
`note`	Notable events	Important but non-critical events
`ok`	Success confirmation	Health checks, validations
`success`	Major success	Completed operations
`warn`	Warnings	Degraded performance, approaching limits
`error`	Errors	Failures requiring attention
`lifecycle`	Lifecycle events	Start, stop, restart, deploy

License and Legal Information

Trademarks

Company Information

Task Venture Capital GmbH Registered at District Court Bremen HRB 35230 HB, Germany

For any legal inquiries or further information, please contact us via email at hello@task.vc.

By using this repository, you acknowledge that you have read this section, agree to comply with its terms, and understand that the licensing of the code does not imply endorsement by Task Venture Capital GmbH of any derivative works.

changelog.md for @push.rocks/smartlog

2026-02-20 - 3.2.1 - fix(destination-buffer)

return entries in chronological order (oldest-first) and adjust pagination semantics

- Change getEntries to return the most recent entries in chronological (oldest-first) order instead of newest-first
- Adjust pagination to compute slice indices from the newest end (start = max(0, len - limit - offset), end = len - offset)
- Update tests to expect chronological ordering and clarified pagination examples
- Modified files: ts_destination_buffer/classes.destinationbuffer.ts, test/test.destination-buffer.node.ts

2026-02-19 - 3.2.0 - feat(destination-buffer)

add SmartlogDestinationBuffer in-memory circular buffer destination with query/filter/pagination and tests

- Introduce SmartlogDestinationBuffer: an in-memory circular buffer implementing ILogDestination with configurable maxEntries (default 2000).
- Expose APIs: handleLog, getEntries (supports level filtering, search, since timestamp, limit/offset pagination, newest-first ordering), getEntryCount, and clear.
- Add package export './destination-buffer' and module entry points (index and plugins) to expose the new destination.
- Add tests covering storage, filtering by level and search, pagination, eviction when maxEntries exceeded, clearing, and default maxEntries behavior.

2026-02-14 - 3.1.11 - fix(destination-receiver)

return full webrequest response from SmartlogDestinationReceiver and migrate to WebrequestClient; update tests, dependencies, docs, and npmextra metadata

- SmartlogDestinationReceiver now uses plugins.webrequest.WebrequestClient instead of plugins.webrequest.WebRequest and returns the full response object instead of response.body — callers expecting the previous shape will need to adapt (breaking change)
- Tests updated to match the new webrequest.postJson return shape
- Bumped several devDependencies and dependencies (@git.zone/* toolchain, @push.rocks/* libs) to newer major/minor versions
- Removed tsbundle from the build script and adjusted build tooling invocation
- Added pnpm.onlyBuiltDependencies and updated npmextra.json namespace keys and release/tsdoc metadata
- Documentation (readme.md) updated: examples changed to async/await usage, expanded legal and issue-reporting sections

2025-09-22 - 3.1.10 - fix(tests)

Bump dependency versions and adjust test to use enableConsole() default

- Update devDependencies: @git.zone/tsbuild -> ^2.6.8, @git.zone/tsbundle -> ^2.5.1, @git.zone/tstest -> ^2.3.6
- Update runtime dependencies: @push.rocks/consolecolor -> ^2.0.3, @push.rocks/smartfile -> ^11.2.7, @push.rocks/smarthash -> ^3.2.3
- Simplify test invocation in test/test.ts: call testSmartLog.enableConsole() without the captureAll option

2025-09-01 - 3.1.9 - fix(docs)

Update README: expand documentation, examples and usage guides

- Fully rewrote README to a comprehensive documentation page with badges, motivation and feature overview.

- Added Quick Start, detailed usage examples and code snippets for Smartlog, destinations, and custom destinations.
- Documented interactive console features (spinners, progress bars) including non-interactive fallbacks and configuration options.
- Expanded sections on built-in destinations (console, file, devtools, ClickHouse, receiver) with practical examples and env-driven configuration.
- Added integration examples (PM2, Docker), CLI usage, advanced features (context management, scoped logging, minimum log levels) and best practices.
- Included API reference pointers and contributing / license information.

2025-05-20 - 3.1.8 - fix(devDependencies)

Update devDependencies for tstest and Node types

- Bump @git.zone/tstest from ^1.7.0 to ^1.9.0
- Bump @types/node from ^22.15.18 to ^22.15.20

2025-05-20 - 3.1.7 - fix(ts_destination_local)

Update debug log color: set textColor to 'pink' in DestinationLocal.

- Changed debug log text color from 'gray' to 'pink' for improved consistency in log output

2025-05-20 - 3.1.6 - fix(ts_destination_local)

Update debug prefix color in DestinationLocal: change from gray to pink for improved visibility.

- Adjusted the 'debug' log prefix color in classes.destinationlocal.ts to use 'pink' instead of 'gray'.

2025-05-20 - 3.1.5 - fix(core)

Maintain and verify project metadata and commit info consistency

- No code changes; confirming commit info files and documentation remain aligned
- Ensured consistent versioning across submodules and package metadata

2025-05-20 - 3.1.4 - fix(DestinationLocal)

Fix debug log rendering, add fallback for unknown log levels, and correct error prefix typo in local destination

- Added tests for debug and unknown log levels in DestinationLocal
- Refactored log string generation to use a fallback style for undefined levels
- Fixed typo: replaced non-existent 'errorPrefix' with 'error.prefix'

2025-05-19 - 3.1.3 - fix(documentation)

Update API reference and enhance README with detailed examples and usage instructions

- Added comprehensive API.md with reference details for core modules, interfaces, and interactive console features
- Updated README.md with improved installation steps, usage examples, and log group information
- Revised development hints to reflect environment detection and test file organization

2025-05-16 - 3.1.2 - fix(tests)

Update test imports and devDependencies to use @git.zone/tstest/tapbundle

- Changed import statements in test files from '@push.rocks/tapbundle' to '@git.zone/tstest/tapbundle'

- Updated devDependency '@git.zone/tstest' to version ^1.7.0 and removed dependency on '@push.rocks/tapbundle'

2025-05-15 - 3.1.1 - fix(source-interactive)

Fix import path in receiver tests and rename progress bar property for clarity; update SmartlogSourceOra getter for improved backward compatibility.

- Changed test file import from '../ts/index.js' to '../dist_ts/index.js' in test.receiver.node.ts to resolve module path issues
- Renamed property 'complete' to 'completeChar' in SmartlogProgressBar and updated its usage accordingly
- Modified SmartlogSourceOra getter to use public methods for starting and stopping the spinner, ensuring backward compatibility

2025-05-15 - 3.1.0 - feat(interactive)

Add interactive console features and refactor spinner module by renaming source-ora to source-interactive and removing ora dependency

- Renamed source-ora module to source-interactive and updated package.json exports
- Removed ora dependency in favor of a custom spinner implementation
- Added new progress bar functionality with configurable options including ETA, percentage, and color
- Updated tests and documentation (README and plan) to reflect the new interactive features
- Bumped dependency versions in package.json and improved test script configuration

2025-05-12 - 3.0.9 - fix(test/destination-

devtools.browser)

Simplify DevTools browser tests by removing redundant styled log assertions.

- Removed detailed log styling tests to streamline the browser test suite.
- Retained a basic test to verify DevTools destination instance creation.

2025-05-11 - 3.0.8 - fix(ci)

Update CI workflows, build scripts, and export configuration

- Add separate Gitea workflows for tag and non-tag pushes to automate testing, auditing, and release steps
- Revise package.json exports and update dependency versions and build scripts for improved module resolution
- Enhance tsconfig settings with baseUrl and paths for consistency
- Refine source code formatting and adjust test cases for better maintenance

2024-06-06 - 3.0.7 - no notable changes

This release contains no detailed changes.

2024-06-06 - 3.0.6 - core

A few fixes and updates were made in this release.

- fix(core): update
- update description

2024-05-17 - 3.0.5 - core

A minor core fix was applied.

- fix(core): update

2024-05-17 - 3.0.4 - core

This release includes several updates to core files and configuration.

- fix(core): update
- update tsconfig
- update npmextra.json: githost
- update npmextra.json: githost
- update npmextra.json: githost

2023-08-08 - 3.0.3 - core

A simple core update was performed.

- fix(core): update

2023-07-12 - 3.0.2 - core

Core updates and a change of organizational scheme were introduced.

- fix(core): update
- switch to new org scheme (x2)

2022-10-26 - 3.0.1 - core

A core fix was applied.

- fix(core): update

2022-07-26 - 3.0.0 - core

A minor core update was made.

- fix(core): update

2022-06-26 - 2.0.44 - core

A breaking change switched the module system to esm.

- BREAKING CHANGE(core): switch to esm

2021-07-21 - 2.0.43 - core

A core update was applied.

- fix(core): update

2021-07-20 - 2.0.42 - core

A core update was applied.

- fix(core): update

2021-07-20 - 2.0.41 - core

A core update was applied.

- fix(core): update

2021-07-20 - 2.0.40 - core

A core update was applied.

- fix(core): update

2021-07-06 - 2.0.39 - core

A core update was applied.

- fix(core): update

2020-09-08 - 2.0.38 - core

A core update was applied.

- fix(core): update

2020-09-07 - 2.0.37 - core

A core update was applied.

- fix(core): update

2020-09-07 - 2.0.36 - core

A core update was applied.

- fix(core): update

2020-08-02 - 2.0.35 - core

A core update was applied.

- fix(core): update

2020-06-11 - 2.0.34 - core

A core update was applied.

- fix(core): update

2020-06-11 - 2.0.33 - core

A core update was applied.

- fix(core): update

2020-06-11 - 2.0.32 - core

A core update was applied.

- fix(core): update

2020-06-11 - 2.0.31 - core

A core update was applied.

- fix(core): update

2020-06-11 - 2.0.30 - core

A core update was applied.

- fix(core): update

2020-06-11 - 2.0.29 - core

A core update was applied.

- fix(core): update

2020-06-11 - 2.0.28 - core

A core update was applied.

- fix(core): update

2020-06-08 - 2.0.27 - core

A core update was applied.

- fix(core): update

2020-06-08 - 2.0.26 - core

A core update was applied.

- fix(core): update

2020-06-08 - 2.0.25 - core

A core update was applied.

- fix(core): update

2020-06-08 - 2.0.24 - core

A core update was applied.

- fix(core): update

2020-06-07 - 2.0.23 - core

A core update was applied.

- fix(core): update

2020-06-05 - 2.0.22 - core

A core update was applied.

- fix(core): update

2020-06-05 - 2.0.21 - core

A core update was applied.

- fix(core): update

2019-10-22 - 2.0.20 - core

A core update was applied.

- fix(core): update

2019-10-22 - 2.0.19 - core

Dependencies were updated in the core.

- fix(core): update dependencies

2019-01-30 - 2.0.18 - license

License files were updated.

- fix(license): update license files

2019-01-30 - 2.0.17 - readme

A typo in the readme was fixed.

- fix(readme): fix typo

2019-01-30 - 2.0.16 - readme

A typo in the readme was fixed.

- fix(readme): fix typo

2019-01-28 - 2.0.15 - core

A core update was applied.

- fix(core): update

2019-01-28 - 2.0.14 - core

A core update was applied.

- fix(core): update

2019-01-22 - 2.0.13 - core

A core update was applied.

- fix(core): update

2019-01-22 - 2.0.12 - core

A core update was applied.

- fix(core): update

2019-01-18 - 2.0.11 - core

A core update was applied.

- fix(core): update

2019-01-15 - 2.0.10 - core

A core update was applied.

- fix(core): update

2019-01-15 - 2.0.9 - core

A core update was applied.

- fix(core): update

2018-11-11 - 2.0.8 - core

A core update was applied.

- fix(core): update

2018-11-04 - 2.0.7 - core

A core update was applied.

- fix(core): update

2018-11-04 - 2.0.6 - core

A core update was applied.

- fix(core): update

2018-11-04 - 2.0.5 - api

An API improvement was made.

- fix(api): streamline api

2018-11-04 - 2.0.4 - core

A core update was applied.

- fix(core): update

2018-11-03 - 2.0.3 - core

A core update was applied.

- fix(core): update

2018-10-31 - 2.0.2 - core

A core update was applied.

- fix(core): update

2018-10-30 - 2.0.1 - core

A new log router was implemented in core.

- fix(core): implement log router

2018-07-10 - 2.0.0 - core

Console enabling was fixed.

- fix(.enableConsole()): now works

2018-07-10 - 1.0.6 - core

A breaking change added a simple defaultLogger to core.

- BREAKING CHANGE(core): now has simple defaultLogger

2018-07-08 - 1.0.5 - npm

Package distribution was fixed.

- fix(npm): package distribution

2018-06-05 - 1.0.4 - package

The package name was updated.

- fix(package): update package name

2018-06-05 - 1.0.3 - core

Several changes were made to update standards and system behavior.

- fix(core): update to latest standards
- system change

2018-01-28 - 1.0.2 - docs

Documentation was added to the readme.

- docs(readme): add readme

2018-01-28 - 1.0.1 - misc

Initial cleanup and TypeScript support were introduced.

- fix(cleanup):
- feat(ts): initial

2016-10-15 - 1.0.0 - no notable changes

This initial release contains no detailed changes.