readme.md for @signature.digital/tools

A comprehensive TypeScript library for **digital contract management** – covering every stage from drafting and collaboration to e-signatures, legal compliance, and archival. □□□

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.

Features at a Glance

□□ **99.9% Contract Coverage** – Supports employment, service, sales, lease, license, NDA, financial, and dozens more contract types □□ **eIDAS Compliant Signatures** – Simple, advanced, and qualified electronic signatures with full audit trails □□ **Identity Verification** – Passport/NFC, document+selfie, video ident, and third-party IdP support □□ **Legal Compliance** – RFC 3161 TSA timestamps, blockchain anchoring, LTV (Long-Term Validation) □□ **Version Control** – Semantic versioning, branches, diffs, and amendment tracking □□ **Real-time Collaboration** – Comments, suggestions, presence tracking, conflict resolution □□ **Complete Audit Logs** – Hash-chained, tamper-evident action history □□ **PDF Generation** – Configurable templates, branding, security, and digital signing □□ **TypeScript-First** – Full type safety with comprehensive interfaces and factory functions

Install

```
# Using pnpm (recommended)
pnpm add @signature.digital/tools
```

```
# Using npm
npm install @signature.digital/tools
```

Quick Start

```
import {
  createEmptyContract,
  createRole,
  createInvolvedParty,
  createParagraph,
  type IPortableContract,
} from '@signature.digital/tools';

// Create a new employment contract
const contract = createEmptyContract(
  'Software Developer Employment Agreement',
  'employment',
  'employment_full_time',
  'user-123',
  'en'
);

// Define roles
const employerRole = createRole('employer', 'Employer', 'The hiring company');
const employeeRole = createRole('employee', 'Employee', 'The hired individual');

contract.availableRoles.push(employerRole, employeeRole);

// Add contract content
contract.paragraphs.push(
  createParagraph('Scope of Work', 'The Employee shall perform software development
duties...', 'clause', 1),
  createParagraph('Compensation', 'The Employee shall receive a monthly salary of...',
'clause', 2)
);

console.log(contract.id); // UUID
```

```
console.log(contract.lifecycle.currentStatus); // 'draft'
```

Package Structure

This package provides **three entry points**:

Entry Point	Package Name	Description
`@signature.digital/tools`	`@signature.digital/tools`	Main export – re-exports all interfaces
`@signature.digital/tools/interfaces`	`@signature.digital/interfaces`	Core TypeScript interfaces
`@signature.digital/tools/demodata`	`@signature.digital/demodata`	Demo contracts for testing

Core Concepts

□□ Contract Types

The library supports a comprehensive taxonomy of contract types:

```
type TContractCategory =
  | 'employment'    // Full-time, part-time, minijob, freelance, executive...
  | 'service'       // SLA, MSA, maintenance, support agreements
  | 'sales'         // Purchase orders, distribution, supply agreements
  | 'lease'         // Residential, commercial, equipment, vehicle
  | 'license'       // Software, patent, trademark, franchise
  | 'partnership'   // Joint ventures, strategic alliances
  | 'confidentiality' // NDA (unilateral/bilateral/multilateral)
  | 'financial'     // Loans, guarantees, investments
  | 'other';        // Settlement, POA, LOI, MOU
```

□□ Signature System

Full support for eIDAS-compliant signatures:

```
import type { ISignature, TSignatureType, TSignatureLegalLevel } from
'@signature.digital/tools';
```

```
// Supported signature types
type TSignatureType = 'drawn' | 'typed' | 'click' | 'digital' | 'qualified' | 'biometric';

// eIDAS legal levels
type TSignatureLegalLevel = 'simple' | 'advanced' | 'qualified';
```

Signature data structures include:

- **Drawn signatures** – Stroke data from signature pads (compatible with signature_pad library)
- **Typed signatures** – Text with font styling
- **Click-to-sign** – Acknowledgment-based acceptance
- **Digital signatures** – X.509/PKI certificate-based (PKCS#7)
- **Qualified signatures** – eIDAS QES via QTSP
- **Biometric** – Fingerprint, facial, voice (extensible)

□□ Identity Verification

```
import type { IIdentityVerificationResult, TIdentityVerificationMethod } from
'@signature.digital/tools';

// Supported verification methods
type TIdentityVerificationMethod =
  | 'email' | 'sms' | 'knowledge'
  | 'document_upload' | 'document_nfc' | 'document_ocr'
  | 'biometric_facial' | 'video_ident'
  | 'bankid' | 'eid' | 'third_party_idp';
```

⚖ Legal Compliance

RFC 3161 TSA Timestamps:

```
import type { ITsaTimestamp, IBlockchainTimestamp } from '@signature.digital/tools';

// TSA timestamp with qualified status
interface ITsaTimestamp {
  authority: ITsaAuthority;
  token: ITsaToken;
  verification: ITsaVerification;
```

```
  qualifiedInfo?: IQualifiedTsaInfo;
}
```

Blockchain Anchoring:

```
// Supported networks
type TBlockchainNetwork = 'bitcoin' | 'ethereum' | 'polygon' | 'arbitrum' | 'optimism' |
'hyperledger' | 'private';
```

□□ Contract Content

Paragraphs support rich features:

```
import type { IParagraph, IVariable, ICondition } from '@signature.digital/tools';

// Section types
type TSectionType = 'preamble' | 'definitions' | 'clause' | 'subclause' | 'schedule' |
'exhibit' | 'annex' | 'amendment' | 'signature_block' | 'witness_block' | 'acknowledgment' |
'recital';

// Variables with validation and formatting
interface IVariable {
  variableId: string;
  name: string;
  type: TVariableType; // 'text' | 'number' | 'currency' | 'date' | 'party_name' |
'calculated'...
  value?: unknown;
  required: boolean;
  validation?: IVariableValidation;
  format?: IVariableFormat;
}
```

□□ Financial Terms

Machine-readable financial data:

```
import type { IFinancialTerms, IPaymentScheduleEntry } from '@signature.digital/tools';
```

```
const financialTerms: IFinancialTerms = {
  totalValue: { amount: 50000, currency: 'EUR', includesTax: false },
  paymentSchedule: [...],
  paymentMethods: ['bank_transfer', 'sepa_direct_debit'],
  billingRates: [...],
  penalties: [...],
};
```

□□ Time Terms

Duration, milestones, renewal, and termination:

```
import type { ITimeTerms, IRenewalTerms, ITerminationTerms } from '@signature.digital/tools';

const timeTerms: ITimeTerms = {
  effectiveDate: Date.now(),
  duration: { value: 12, unit: 'months' },
  isIndefinite: false,
  renewal: {
    type: 'auto_renew',
    renewalPeriod: { value: 12, unit: 'months' },
    maxRenewals: 3,
  },
  termination: {
    noticePeriod: { duration: { value: 30, unit: 'days' }, form: 'written' },
  },
};
```

□□ Version Control

Git-like versioning for contracts:

```
import { createInitialVersion, incrementVersion, versionToString } from
'@signature.digital/tools';

const v1 = createInitialVersion('user-123');
const v2version = incrementVersion(v1.version, 'minor');
console.log(versionToString(v2version)); // '0.2.0'
```

□□ Collaboration

Real-time collaboration support:

```
import type { ICommentThread, ISuggestion, IUserPresence } from '@signature.digital/tools';
import { createCommentThread, createSuggestion, createUserPresence } from
'@signature.digital/tools';

// Track user presence
const presence = createUserPresence('user-123', 'Alice', '#1a73e8');

// Comments with threads
const thread = createCommentThread(contractId, versionId, {
  type: 'text_range',
  paragraphId: 'para-1',
  textRange: { start: 0, end: 50, quotedText: 'Sample text' },
}, 'user-123');

// Track-changes style suggestions
const suggestion = createSuggestion(
  contractId, versionId, 'para-1',
  'replace', 'old text', 'new text',
  'user-456', 'Bob'
);
```

□□ Audit Logging

Tamper-evident, hash-chained audit logs:

```
import type { IAuditLog, IContractAction, TActionCategory } from '@signature.digital/tools';

// Action categories
type TActionCategory =
  | 'create' | 'view' | 'edit' | 'status_change'
  | 'share' | 'permission_change' | 'comment'
  | 'signature_request' | 'signature_given' | 'signature_declined'
  | 'export' | 'print' | 'download'
  | 'archive' | 'restore' | 'delete';
```

□□ PDF Generation

Comprehensive PDF configuration:

```
import { createDefaultPdfConfig, type IPdfGenerationConfig } from '@signature.digital/tools';

const pdfConfig = createDefaultPdfConfig('Employment Contract');

// Customize branding
pdfConfig.branding.primaryColor = '#1a73e8';
pdfConfig.branding.logo = {
  url: 'https://example.com/logo.png',
  position: 'left',
  maxWidth: 150,
  maxHeight: 50,
};

// Security settings
pdfConfig.security = {
  encryption: { enabled: true, algorithm: 'AES-256' },
  permissions: {
    printing: 'high_resolution',
    copying: false,
    modifying: false,
  },
};
```

Demo Data

Test your implementation with realistic demo contracts:

```
import { demoContract } from '@signature.digital/tools/demodata';

console.log(demoContract.title); // 'Minijob Employment Contract'
console.log(demoContract.metadata.governingLaw.country); // 'DE'
console.log(demoContract.paragraphs.length); // 8 fully structured paragraphs
```

The demo contract showcases:

- Multi-language support (DE/EN)
- Variable placeholders with validation
- Legal references (BGB, SGB IV, MiLoG)
- Financial terms with payment schedules
- Company and person contacts with full details
- Proper role and party structure

Factory Functions

The library provides factory functions for creating properly initialized objects:

```
// Contract creation
createEmptyContract(title, category, contractType, createdBy, language?)
createDefaultMetadata(category, contractType, language?)
createDefaultGoverningLaw()

// Roles and parties
createRole(id, name, description, options?)
createInvolvedParty(roleId, contact, signingOrder?)

// Content
createParagraph(title, content, sectionType?, order?)
createVariable(variableId, name, type, required?)

// Terms
createEmptyFinancialTerms()
createEmptyTimeTerms()
createEmptyObligationTerms()

// Versioning
createInitialVersion(userId, userDisplayName?)
createEmptyVersionHistory(contractId, initialVersion)
incrementVersion(current, incrementType)
versionToString(version)
parseVersionString(versionString)

// Collaboration
createCommentThread(contractId, versionId, anchor, userId)
```

```
createComment(threadId, authorId, authorDisplayName, content)
createSuggestion(...)
createCollaborator(userId, contact, permission, invitedBy)
createUserPresence(userId, displayName, color)

// Lifecycle
createInitialLifecycle(contractId)
createEmptyAuditLog(contractId)
createStatusTransition(fromStatus, toStatus, triggeredBy, reason?)
createLegalHold(name, description, contractIds, createdBy, reason)

// Attachments
createContractAttachment(contractId, versionId, type, category, title, addedBy)
createPriorContractReference(relationshipType, contractId?, externalReference?)
createDocumentBundle(name, mainContractId, purpose, createdBy)
createAttachmentFile(filename, mimeType, size, storageProvider, storageKey, checksum)

// Identity verification
createIdentityVerificationRequest(methods, requiredConfidence, expiresInSeconds?)
createPendingVerificationResult(requestId)

// Legal compliance
createEmptyLegalComplianceProof()
createPendingTsaTimestamp(authorityUrl)
createPendingBlockchainTimestamp(network, dataHash)

// PDF
createDefaultPdfConfig(title)
createDefaultBranding()
createDefaultLayout()
createDefaultContentOptions()
createDefaultPageNumbering()

// Signatures
createEmptySignatureMetadata()
createDefaultSignatureFieldRequirements()
```

TypeScript Integration

All interfaces follow strict naming conventions:

- **Interfaces**: `I` prefix (e.g., `IPortableContract`)
- **Types**: `T` prefix (e.g., `TContractStatus`)

```
import type {
  // Core interfaces
  IPortableContract,
  IRole,
  IInvolvedParty,
  IParagraph,

  // Signature system
  ISignature,
  ISignatureField,
  TSignatureType,
  TSignatureLegalLevel,

  // Identity
  IIdentityVerificationResult,
  TIdentityVerificationMethod,

  // Legal
  ILegalComplianceProof,
  ITsaTimestamp,
  IBlockchainTimestamp,

  // Terms
  IFinancialTerms,
  ITimeTerms,
  IObligationTerms,

  // And 200+ more types...
} from '@signature.digital/tools';
```

License and Legal Information

This repository contains open-source code licensed under the MIT License. A copy of the license can be found in the LICENSE file.

Please note: The MIT License does not grant permission to use the trade names, trademarks, service marks, or product names of the project, except as required for reasonable and customary use in describing the origin of the work and reproducing the content of the NOTICE file.

Trademarks

This project is owned and maintained by Task Venture Capital GmbH. The names and logos associated with Task Venture Capital GmbH and any related products or services are trademarks of Task Venture Capital GmbH or third parties, and are not included within the scope of the MIT license granted herein.

Use of these trademarks must comply with Task Venture Capital GmbH's Trademark Guidelines or the guidelines of the respective third-party owners, and any usage must be approved in writing. Third-party trademarks used herein are the property of their respective owners and used only in a descriptive manner, e.g. for an implementation of an API or similar.

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.

Revision #3
Created 2026-03-28 11:14:48 UTC by foss.global Team
Updated 2026-03-28 12:21:35 UTC by foss.global Team