

# Sign Live! cloud suite gears fsm manual

Dezember 2025

intarsys GmbH

Sign Live! cloud suite gears fsm manual

Version 1.1

cloud suite fsm architecture, design & reference

intarsys GmbH  
Sign Live! cloud suite gears fsm manual  
Version 1.1

All rights reserved  
© 2024 intarsys GmbH  
[www.intarsys.de](http://www.intarsys.de)



# Preface

---

- Author and company

This book has been provided by different authors from the development staff of intarsys GmbH.

- Trademarks

Wherever possible and where the authors were aware of a trademark claim, such designations are marked as trademarks in this book.

jPod is a trademark of intarsys consulting.

Sun, Java and JavaScript are trademarks of Oracle

Microsoft and Windows are trademarks of Microsoft Corporation.

- Who should read this book

This book provides both an overview of the product design and architecture and a reference for using the components and services.

So, this is the document for architects, developers and operators.

- Reviews and comments

We make constant efforts to improve our documentation and meet your requirements. Your comments are welcome and are a valuable resource for us.

E-Mail [support@intarsys.de](mailto:support@intarsys.de)

Website [www.intarsys.de](http://www.intarsys.de)

## ■ Disclaimer

Every effort has been made to make this book as complete and accurate as possible, but no warranty is implied.

The information is provided “as is”. The authors shall in no way be liable to any person or entity with respect to any loss or damages arising from the information contained in this book, or from the use of the disks or programs that may accompany it.

# Contents

Preface	5
▪ Author and company	5
▪ Trademarks	5
▪ Who should read this book	5
▪ Reviews and comments	5
▪ Disclaimer	6
Contents	7
Introduction	10
1. Overview	11
2. Installation	12
2.1 Overview	12
2.2 System requirements	12
2.2.1 Server	12
2.2.2 Client	14
2.3 Compatibility with Sign Live! cloud suite products	14
2.3.1 Sign Live! cloud suite gears	14
2.3.2 Sign Live! cloud suite validator	14
2.4 Documentation	14
2.5 Installation Process	14
2.6 Web App	14
2.7 Version check	15
2.7.1 In the browser log	15
2.7.2 In the WEB-INF directory	15
2.7.3 In the log	15
3. Quick-Start Guide	16
4. Monitor	17
4.1 Overview	17
4.2 Configuration	17
4.2.1 Monitor Configuration Location	17

## Content

4.2.2	Monitor Configuration	17
4.2.3	Service Configuration	22
5.	Web UI	25
5.1	Browser security	25
5.1.1	Headers	25
5.1.2	Header configuration	27
6.	Authentication	28
6.1	Overview	28
6.2	Opt-out	28
6.3	Security Overview	28
6.4	Logout	29
6.5	Authentication filter	29
6.5.1	Static authentication	29
6.5.2	Basic authentication	30
6.5.3	OAuth2 authentication	30
6.6	Authentication manager	31
6.6.1	In-memory repository	31
6.7	Well known security beans	32
7.	Configuration	33
7.1	Overview	33
7.1.1	Configuration location	33
7.1.2	Built-in configuration	35
7.1.3	Custom property definition	35
7.1.4	Custom bean definition	35
7.1.5	Using profiles	36
7.1.6	String expansion integration	36
7.2	Web application root	37
7.3	Logging	37
7.3.1	Override logging	37
7.3.2	Built-in logging	38
7.3.3	Log tweaks	39
7.4	Licenses	39
8.	String expansion	41
8.1	Basics	41
8.1.1	Why string expansion	41
8.1.2	Terminology	41
8.1.3	Syntax	42
8.1.4	Constant text in expression	42
8.1.5	Namespaces	42
8.1.6	Spring integration	43
8.2	Namespaces	44
8.2.1	Overview	44
8.2.2	app	44
8.2.3	config	44



## Content

8.2.4	counters	45
8.2.5	entity	45
8.2.6	environment	46
8.2.7	system	47
8.2.8	time	48
8.2.9	file	48
8.2.10	event	49
8.3	Formatting	50
8.3.1	Overview	50
8.3.2	String formatting	51
8.3.3	Integer formatting	52
8.3.4	Date formatting	53
8.3.5	Float formatting	54
8.3.6	File path formatting	55
8.3.7	Default value	56
8.3.8	Recursion	56
8.3.9	Conditional evaluation	57
9.	Appendices	59
9.1	Cheat sheet	59
9.1.1	Windows locations	59
9.1.2	Linux locations	59
10.	External References	60

# Introduction

---

Sign Live! cloud suite gears fsm provides a modern file system monitoring (fsm) solution which complements the 'Sign Live! cloud suite' product line.

This book gives an overview of the architecture and design decisions for Sign Live! cloud suite gears fsm, along with a detailed reference of how to configure and use.

# 1. Overview

---

Sign Live! cloud suite gears fsm is a file system monitoring (fsm) solution designed for seamless bulk processing within the Sign Live! cloud suite product family. With its web-based user interface users can configure settings and monitor file system activities.

Sign Live! cloud suite gears fsm enables automatic processing of files placed in the input directory by integrating with services such as gears or validator. For example, users can define input and output directories, allowing files moved to the input directory to be automatically processed by services like gears or validator. After processing the files are then transferred to the output directory.

This solution simplifies bulk file processing and enhances productivity by working in tandem with other Sign Live! cloud suite products.

## 2. Installation

### 2.1 Overview

The product is shipped in a ZIP or TAR file, containing

- doc
- example
- webapps

### 2.2 System requirements

#### 2.2.1 Server

##### 2.2.1.1 Operating system

The server components are tested and released for

- Ubuntu Linux 24.04.3 LTS
- Microsoft Windows 2016 / 2019 / 2022 / 2025 server

##### 2.2.1.2 System software

##### 2.2.1.2.1 **Java server runtime**

The server components are developed in Java and require

- Java Runtime Environment 17 (64 bit)

The software is tested and released using Azul Zulu OpenJDK 17.60.17.

##### 2.2.1.2.2 **Servlet container Tomcat**

The server components are deployed as "war" files. The server components are tested and released using

- Tomcat 10.1.48 (64 bit)

For server side rendering the Java VM is switched to headless mode automatically (-Djava.awt.headless=true).

##### 2.2.1.2.3 **Reverse proxy**

In complex installations you should think about installing Sign Live! cloud suite gears fsm behind a reverse proxy (like nginx).

This may ease maintenance when it comes to SSL/TLS termination.

### 2.2.1.3 Hardware

#### 2.2.1.3.1 CPU

The system is tested on

- x86-64

architecture only.

For normal operation we recommend at least

- medium sized recent Core i5 or equivalent

Depending on the number of users and system setup (e.g. encrypted file repository, using server-side rendering) you may want to increase server power.

#### 2.2.1.3.2 Memory

Memory requirements depend heavily on the number of concurrent requests. We recommend at least

- 4 GB RAM

Depending on the number of users and system setup (e.g. encrypted file repository, using server-side rendering) you may want to increase available memory.

#### 2.2.1.3.3 Disk storage

Beside the software installation requirements

- ~ 1GB (Tomcat, Java, Sign Live! cloud suite gears fsm components)

You need space for the server-side repository if documents are stored temporarily on the server. This depends on the number of users, frequency and duration of requests.

- ~ 10GB

should be enough in small to medium sized installations.

As described in chapter 7.3.2 the built-in logging feature leads to the creation of log files. The total space usage is capped globally at

- 2GB.

#### 2.2.1.3.4 Peripherals

The system components do not require special peripherals.

## 2.2.2 Client

### 2.2.2.1 Web browser

Some components require a browser frontend. This is implemented using HTML5 and modern JavaScript features. The following browsers are tested and supported.

- Chrome > 139 (Windows 10/11)
- Firefox > 142 (Windows 10/11)
- Edge Chromium > 140 (Windows 10/11)

## 2.3 Compatibility with Sign Live! cloud suite products

### 2.3.1 Sign Live! cloud suite gears

Version 8.13.1 or newer are tested and supported.

### 2.3.2 Sign Live! cloud suite validator

Version 8.13.2 or newer are tested and supported.

## 2.4 Documentation

The documentation is contained in the "doc" folder of the deployment.

## 2.5 Installation Process

Perform the following installation steps:

1. Install Java Runtime
2. Install Tomcat
3. Deploy the "cloudsuite-gears#fsm.war" into the Tomcat webapps directory.
4. Start the tomcat application
5. Open the following URL in the browser  
**<http://<host>:8080/cloudsuite-gears/fsm>**

## 2.6 Web App

You can install the web app by simply deploying the "cloudsuite-gears#fsm.war" to a standard servlet container.

This will deploy the default product configuration.

If you need further refinements, see the section "Configuration" of this manual.

The web application needs write access to the `${cloudsuite.data.shared}` directory. This means that for a Linux installation, you must create this directory and grant access to the user running the servlet container.

## 2.7 Version check

### 2.7.1 In the browser log

The Sign Live! cloud suite gears fsm web app will print a version signature in the browser log upon start up:

```
cloud suite gears fsm v.1.0.0, Build 21, 2025-04-23T16:12:13.461Z
```

### 2.7.2 In the WEB-INF directory

In cases you need to physically check the installation version on the deployed server application, you can look up the "`<webapp>/WEB-INF/version.txt`". It contains tags that describe the artifact version

```
version=1.0.0
build=321
timestamp=Mon Apr 23 14:15:54 CEST 2025
```

### 2.7.3 In the log

The log file is the most important source of information for troubleshooting an installation. You can find the version of every component contained in the deployment near the beginning of the log file

```
[09:29:54.009][I][d.i.s.i.PrintVersion][main                ][][] version info:
[09:29:54.009][I][d.i.s.i.PrintVersion][main                ][][] gears-fsm (intarsys-
cloudsuite-gears-fsm-backend-1.0.0.jar), 1.0.0, local, 2025-01-08
```

### 3. Quick-Start Guide

---

This tutorial like section aims to guide you while setting up Sign Live! cloud suite gears fsm for the first time. More details about each step are referenced. In this example the default paths for Windows are mentioned for better readability. For the list of all default paths see chapter 9.1.1.

1. Install and start Sign Live! cloud suite gears fsm (see chapter 2.5).
2. In your browser at `http://<host>:8080/cloudsuite-gears/fsm` you should see the web UI (see chapter 5).
3. Click the button “Add via editor”, enter a name for the new monitor and click the button “Add”.  
The configuration file of this monitor is now saved at  
`%ProgramData%/cloudsuite/data/fsm/monitor/` (see chapter 4.2.1).
4. Start the newly added monitor.
5. Navigate to the root directory of this monitor at  
`%ProgramData%/cloudsuite/data/fsm/work/demoPlain`. Here you find the input, output and error directories.
6. Place a pdf file in the input directory, watch it disappear, and find the signed file in the output directory.
7. Should the file end up in the error directory instead, inspect the log of the monitor in the web UI for more details. Click the “Log” tab next to “Control”.



## 4. Monitor

---

### 4.1 Overview

A monitor in the file system monitoring (fsm) application is responsible for observing a designated directory in the file system, processing specific files, and managing their lifecycle through predefined workflows. It provides file input, output, and error handling.

### 4.2 Configuration

Every monitor has its own configuration file in JSON format. Each configuration consists of two parts: the monitor and the service it uses. See comprehensive examples in the product bundle under “example\demo-monitor-configs”.

#### 4.2.1 Monitor Configuration Location

The monitor configuration files are stored by default in “\${cloudsuite.data.shared}/fsm/monitor”. This can be changed by setting the property “configDirName” in \${cloudsuite.config.shared}/\${cloudsuite.config.name}.properties to another path (see chapter 7.1.1).

#### 4.2.2 Monitor Configuration

The monitor section defines the file handling and lifecycle management of the monitored files. See a simple example below.

## JSON fragment

```

"monitor": {
  "rootDir": "${environment.datadir}/fsm/work/demoPlain",
  "monitorDir": "input",
  "outputDir": "output",
  "errorDir": "error",
  "monitorExtension": ".pdf; .xml",
  "attachmentExtension": ".p7s;.pkcs7;.pkcs#7;.sig;.tsr;.ers;.tags;.xtags",
  "autostart": true,
  "deleteImmediately": true,
  "scanSubDir": true,
  "deleteSubdirectories": false,
  "monitorDelay": 1
}

```

Every supported flag or feature is described in the table below:

rootDir	
String	<p>Default: <code>\${environment.datadir}/fsm/work/&lt;monitor name&gt;</code></p> <p>Root directory, used as the base path for <code>monitorDir</code>, <code>tempDir</code>, <code>outputDir</code>, <code>errorDir</code> unless absolute paths are given for those.</p> <p>Note: <code>\${environment.datadir}</code> = <code>\${cloudsuite.data.shared}</code> = <code>%ProgramData%/cloudsuite/data</code> (windows) or <code>/var/lib/cloudsuite</code> (linux) see chapter 9.1 and chapter 8.2.6.3 (string expansion).</p>
monitorDir	
String	<p>Default: <code>input</code></p> <p>The folder to be observed.</p> <p>An absolute path or a path relative to <code>rootDir</code> can be set.</p>
lockMonitorDir	
Boolean	<p>Default: <code>false</code></p> <p>Lock monitor directory for exclusive monitoring.</p>
scanSubDir	
Boolean	<p>Default: <code>true</code></p> <p>True if sub directories of the input directory should be monitored as well.</p>
deleteSubdirectories	
Boolean	<p>Default: <code>false</code></p> <p>true if empty sub directories are deleted after processing.</p>
deleteImmediately	
Boolean	<p>Default: <code>true</code></p> <p>If the process on the file is synchronous, you should set this property to true, ensuring that the files processed are deleted immediately from the input or temporary directory.</p> <p>If the process performed on the file is asynchronous and you don't get information about its outcome (for example when opening an editor), you can opt for a "lazy" clean up by setting this property to</p>

## Monitor

	false and set “deleteAfter” to some value appropriate for your scenario.
<b>deleteAfter</b>	
Integer	Default: 0 The delay in hours after which the files in the temporary directory are deleted. This is useful only in combination with Delete Immediately=false.
<b>monitorDelay</b>	
Integer	Default: 10 The delay in seconds before the monitor polls again.
<b>testDelay</b>	
Integer	Default: 0 After detecting a file the monitor looks for changes on that file as the writing process may still be active. This is the delay in milliseconds the file monitor waits between two successive tests for the current file size. If the file size or last access timestamp has changed, the file is not yet ready for processing. If the value is 0, no check is made.
<b>monitorExtension</b>	
String	Default: null A list of specific file extensions separated by “;”, like .pdf;.txt. Only files whose names end exactly with one of these extensions are filtered for the monitor. Please keep in mind that a file must be matched by both filters defined by Monitor Extensions and Monitor Patterns.
<b>monitorPattern</b>	
String	Default: null (all files are selected) A list of specific file patterns, like *.pdf;*.txt. This filter allows for “*” and “?” wildcards anywhere in the pattern. Only files that match the given filter are selected. Please keep in mind that a file must be matched by both filters defined by Monitor Extensions and Monitor Patterns.
<b>attachmentExtension</b>	
String	Default: null A list of specific file extensions separated by “;”, like .p7s;.pkcs7;pkcs#7. The monitor processes the files defined by Monitor Extensions and Monitor Patterns, ignoring the files defined by Attachment Extensions. For every accepted input it looks for attachments by adding the specified attachment extension(s) to the input file name resp. to its name without extension. Attachments are moved along with the input file while processing.  Adding a + before a file extension (e.g., +.p7s) triggers behavior where the processing of the master file starts only when the corresponding attachment is also present.
<b>monitorExcludePattern</b>	

String	<p>Default: null</p> <p>Files that match this pattern (e.g. empty*.pdf) are ignored in the monitorDir, remain in the directory and are excluded from attachment processing.</p> <p>In combination with transparentFiles these files could still be moved to the outputDir unprocessed.</p>
<b>packetMarker</b>	
String	<p>Default: null</p> <p>If this property is defined, processing will not start until a file with this name (e.g. marker.txt) is found in a directory. All files in the directory are treated as a packet.</p>
<b>packetGlobalErrorHandling</b>	
Boolean	<p>Default: false</p> <p>If this flag is active, an error while processing a file will result in an error for the whole packet.</p>
<b>packetLogFile</b>	
String	<p>Default: null</p> <p>Example: packet.log</p> <p>If this property is defined, all log entries generated while processing the packet are collected here. The log file is moved together with the packet.</p>
<b>transparentFiles</b>	
Boolean	<p>Default: false</p> <p>If you want to use “transparent pass through”, you must activate this flag.</p> <p>Transparent files are moved by the file system container, but no actions are performed on them. This is useful when we are only a part in a multistep process and the files are targeted at another system.</p>
<b>transparentExtension</b>	
String	<p>Default: null (this property is ignored)</p> <p>A list of specific file extensions separated by “;”, like .pdf;.txt. Only files whose names end exactly with one of these extensions are filtered for the monitor.</p> <p>Please keep in mind that a file must be matched by both filters defined by ~Extensions and ~Patterns.</p>
<b>transparentPattern</b>	
String	<p>Default: null (this property is ignored)</p> <p>A list of specific file patterns, like *.pdf;*.txt. This filter allows for “*” and “?” wildcards anywhere in the pattern. Only files that match the given filter are selected.</p> <p>Please keep in mind that a file must be matched by both filters defined by ~Extensions and ~Patterns.</p>
<b>tempDir</b>	
String	<p>Default: null</p> <p>If defined, a file found in the input is moved to this temporary directory before processing. This is especially useful if the processing</p>

	<p>is asynchronous and the files must be moved to keep from re-processing multiple times.</p> <p>In most cases you will not need to define a temp directory, processing files directly from the input.</p> <p>An absolute path or a path relative to rootDir can be set.</p>
<b>outputDir</b>	
String	<p>Default: output</p> <p>The input file is moved to this directory after processing. The filename used can be configured with Output Filename. The path structure from the input directories are preserved.</p> <p>An absolute path or a path relative to rootDir can be set.</p>
<b>outputFilename</b>	
String	<p>Default: \${event.params.RELATIVEPATH}\${file.name}</p> <p>When Output Directory is defined, the input file is moved. The filename can be configured using this property.</p>
<b>outputFilenameCollision</b>	
String	<p>Default:</p> <p>\${event.params.RELATIVEPATH}\${file.basename}.\${time.uniquemillis:d}.\${file.extension}</p> <p>If the file defined with Output Filename causes a collision, the monitor falls back to this definition.</p>
<b>discardInput</b>	
Boolean	<p>If this flag is set, the input files are simply discarded, not moved to output.</p>
<b>errorDir</b>	
String	<p>Default: error</p> <p>If this property is defined, the complete input is moved to this directory if the processing encounters an error. The path structure from the input directories is preserved.</p> <p>An absolute path or a path relative to rootDir can be set.</p>
<b>errorFilename</b>	
String	<p>Default:</p> <p>\${event.params.RELATIVEPATH}\${event.params.INPUTFILENAME}</p> <p>If Error Directory is defined, the input is moved in case of an error. The filename can be configured using this property.</p>
<b>errorFilenameCollision</b>	
String	<p>Default:</p> <p>\${event.params.RELATIVEPATH}\${file.basename}.\${time.uniquemillis:d}.\${file.extension}</p> <p>If the file defined with Error Filename causes a collision, the monitor falls back to this definition.</p>
<b>autostart</b>	
Boolean	<p>Default: false</p> <p>Automatically starts monitoring when the FSM application launches.</p>
<b>poolSize</b>	
Integer	<p>Default: 1</p>

	Number of threads, allowing several files to be processed simultaneously.
--	---

### 4.2.3 Service Configuration

The service section specifies the type of operation (`implementor`) and its configuration details (`args`). This defines how Sign Live! cloud suite gears fsm interacts with the Sign Live! cloud suite services for tasks such as signing, validating, or timestamping files.

Key	Description
<code>implementor</code>	<p>Defines the service type:</p> <ul style="list-style-type: none"> <li>• <code>SignerService</code></li> <li>• <code>TimestamperService</code></li> <li>• <code>ValidatorService</code></li> </ul> <p>These differ in which Sign Live! cloud suite service is called and how file processing is handled. They call the following endpoints:</p> <ul style="list-style-type: none"> <li>• <a href="https://cloudsuite-intarsys.de/cloudsuite-gears/core/api/v1/flow/signer/create">/cloudsuite-gears/core/api/v1/flow/signer/create</a></li> <li>• <a href="https://cloudsuite-intarsys.de/cloudsuite-gears/core/api/v1/flow/timestampers/create">/cloudsuite-gears/core/api/v1/flow/timestampers/create</a></li> <li>• <a href="https://cloudsuite-intarsys.de/cloudsuite-gears/validator/api/v1/flow/validationpipeline/create">/cloudsuite-gears/validator/api/v1/flow/validationpipeline/create</a></li> </ul>
<code>args</code>	See chapter 4.2.3.1 as well as the example below

#### JSON fragment

```
"service": {
  "implementor": "SignerService",
  "args": {
    "gearsConfig": {
      "baseUrl": "https://cloudsuite.intarsys.de/cloudsuite-gears/core"
    },
    "request": {
      "configuration": "demoPlain"
    }
  }
}
```

#### 4.2.3.1 Args

Key	Description
<code>gearsConfig</code>	Defines how Sign Live! cloud suite gears fsm interacts with the Sign Live! cloudsuite services. Contains server connection settings, authentication mechanisms, and security configurations.

request	Specifies parameters for the service request, such as the target configuration to use.
---------	--

#### 4.2.3.1.1 gearsConfig

In the example below Sign Live! cloudsuite gears has been setup with basic authentication (see “authenticator”) as well as server side tls (“baseUrl” contains https) and client side tls (see “httpConfiguration”). The keystore type is jks (see “keyStoreType”).

##### JSON fragment

```
"gearsConfig": {
  "baseUrl": "https://localhost:8443/cloudsuite-gears/core",
  "httpConfiguration": {
    "hostnameVerifier": "de.intarsys.tools.ssl.StrictHostnameVerifier",
    "connectTimeout": 30000,
    "readTimeout": 30000,
    "useProxy": false,
    "sslContext": {
      "class": "de.intarsys.tools.ssl.ConfigurableSslContextProvider",
      "args": {
        "protocol": "TLS",
        "trustAll": false,
        "sslSessionTimeout": 86400,
        "keyStoreName": "${environment.profiledir}/fsm/certs/demo-client-key.jks",
        "keyStoreType": "JKS",
        "keyStorePassword": "client",
        "keyPassword": "client"
      }
    }
  },
  "authenticator": {
    "class":
"de.intarsys.cloudsuite.gears.core.client.auth.UserPasswordAuthenticator",
    "args": {
      "user": "user",
      "password": "${secret.aes-0#0Yyq+b3cQc/HXx07klqSRiROMYHB6V3x7dgnli6VyCQ=}"
    }
  }
}
```

The following shows an example sslContext with a pkcs12 keystore type.

##### JSON fragment

```
"sslContext": {
  "class": "de.intarsys.tools.ssl.ConfigurableSslContextProvider",
  "args": {
    "protocol": "TLS",
    "trustAll": false,
    "sslSessionTimeout": 86400,
    "keyStoreName": "${environment.profiledir}/fsm/certs/demo-client-key.p12",
    "keyStoreType": "PKCS12",
    "keyStorePassword": "client"
  }
}
```

Key	Description
baseUrl required	This url defines the connection to the Sign Live! cloudsuite gears or validator instance.

httpConfiguration	Here the client TLS can be configured. A client certificate is required (in the example above: <code>demo-client-key.jks</code> )
authenticator	If the Sign Live! cloudsuite service is configured with basic auth security, the required user and password can be entered here. In the example above the password is encrypted via aes. Aes encryption can be done via the Sign Live! cloudsuite gears <a href="#">control panel</a> . If password encryption is used, the Spring string expansion syntax ( <code>\${secret.plain#Zm9v}</code> ) must be used (see e.g. gears manual [1] chapter 8.3.2.2).

#### 4.2.3.1.2 request

This configures the request parameters which are used in the service call made to Sign Live! cloudsuite gears or validator. For further information see the respective API documentation e.g. "RequestSignerCreate"

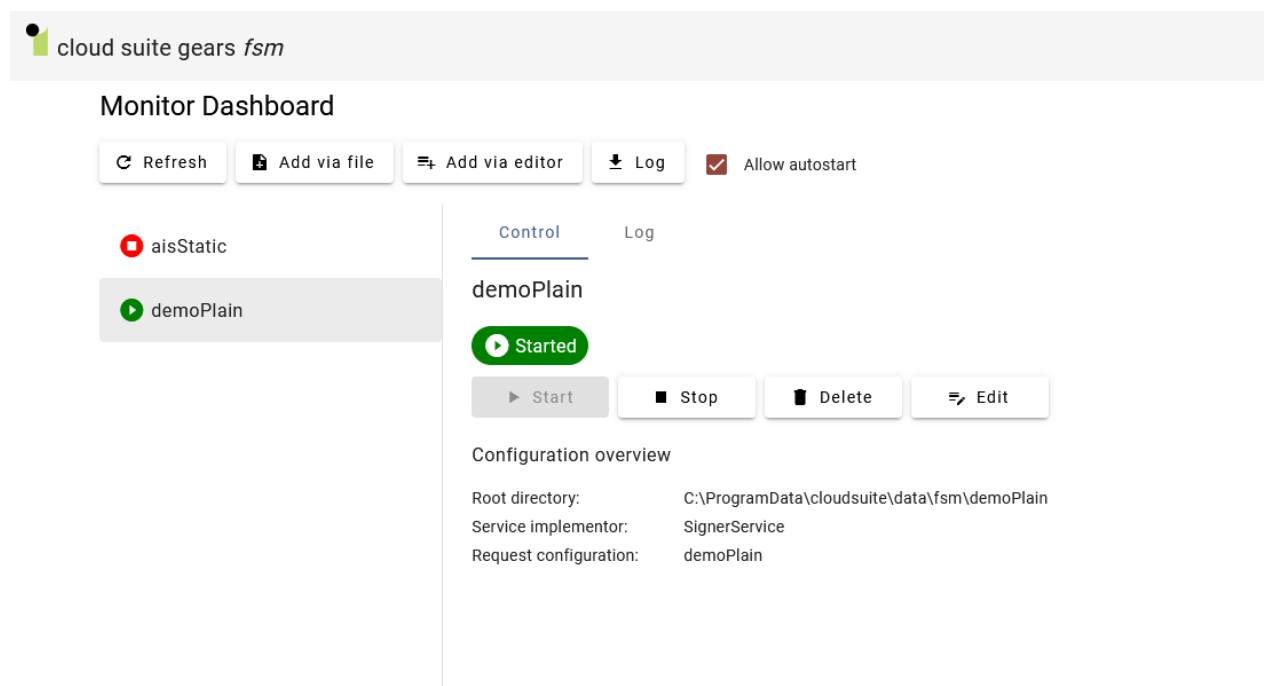
- <https://cloudsuite.intarsys.de/cloudsuite-gears/core/apidoc/index.html#/signer/v1.flow.signer.create>
- <https://cloudsuite.intarsys.de/cloudsuite-gears/validator/apidoc/index.html>



## 5. Web UI

Sign Live! cloud suite gears fsm comes with a web UI. Here users can configure monitor settings and view log entries.

The UI components are quite self-explanatory and provide tooltips. Following the quick-start guide (3) helps as well. All functionality is placed at the “Monitor Dashboard” page.



The web UI can be accessed at <http://<host>:8080/cloudsuite-gears/fsm>.

If authentication (see chapter 6) is set up, a login mask is placed before the entire UI.

### 5.1 Browser security

#### 5.1.1 Headers

The web based UI relies on common browser security features. Sign Live! cloud suite gears fsm will always provide these response headers.

##### 5.1.1.1 referrer-policy

The referrer policy is always set to “no-referrer”.

This is currently not configurable.

## 5.1.1.2 hsts

when HTTPS is requested, the HSTS header is always sent.

This is currently not configurable.

## 5.1.1.3 content-security-policy (CSP)

This technique supersedes the use of many other security headers and is the main means for gears to control browser security.

The default setting is

```
report-uri /cloudsuite-gears/fsm/api/v1/meta/csp/report; report-to default; default-src 'self'; style-src 'self' 'unsafe-inline'; font-src 'self'; img-src 'self' blob: data:; object-src 'none'; frame-ancestors 'none'; connect-src *; worker-src 'self' blob:; script-src-elem 'self' data;;
```

The options in detail

Option	Description
report-uri /cloudsuite-gears/fsm/api/v1/meta/csp/report	All CSP violations are reported back to gears. You can find the entry in the log file.  At the time of writing, the report-uri directive is deprecated and will be replaced by report-to. However, in some browsers it is the only supported way to create CSP reports (e.g. Firefox). Therefore, we use it <b>together</b> with report-to to ensure CSP reports are captured across all browsers.
report-to default	This directive implements the new Reporting API (v1), which uses the Reporting-Endpoints header (see below). All CSP violation reports are sent in groups back to gears. You can find the entry in the log file. Note: report-to only works with HTTPS enabled.
default-src 'self'	Sources may only stem from the page origin
style-src 'self' 'unsafe-inline'	Styles may stem from the page origin *and* may be set inline. This is a requirement of the Angular framework and is considered safe.
font-src 'self'	Fonts must stem from the page source
img-src 'self' blob: data:	Images must stem from the page source or from script evaluations
object-src 'none'	No objects may be embedded
frame-ancestors 'none'	The page itself may not be embedded. This is an options you may need to overwrite when using an embedded gears process.

connect-src *	The page code may connect to any server.
worker-src 'self' blob:	Allows Web Workers only from the page origin or blob: URLs.
script-src-elem 'self' data:	Allows JavaScript elements only from the page origin or data: URLs.

#### 5.1.1.4 Reporting-Endpoints

The header is set to default="/cloudsuite-gears/fsm/api/v1/meta/csp/report".  
This is currently not configurable.

### 5.1.2 Header configuration

For spring internal reasons, you cannot overwrite the bean defining the security realm (as opposed to many other beans). As we consider this a “hotspot” regarding individual integrations, we provide a special property to overwrite the content-security-policy header.

The property is called

```
security.http.securityRealmUI.headers.content-security-policy.directives
```

The value of the property is the complete header content sent to the browser.

Example

```
security.http.securityRealmUICore.headers.content-security-policy.directives= report-uri
/cloudsuite-gears/fsm/api/v1/meta/csp/report; report-to default; default-src 'self';
style-src 'self' 'unsafe-inline'; font-src 'self'; img-src 'self' blob: data;; object-
src 'none'; frame-ancestors 'none'; connect-src *; worker-src 'self' blob;; script-src-
elem 'self' data;;
```

Example, allow embedding. Note the missing “frame-ancestors”

```
security.http.securityRealmUICore.headers.content-security-policy.directives= report-uri
/cloudsuite-gears/fsm/api/v1/meta/csp/report; report-to default; default-src 'self';
style-src 'self' 'unsafe-inline'; font-src 'self'; img-src 'self' blob: data;; object-
src 'none'; connect-src *; worker-src 'self' blob;; script-src-elem 'self' data;;
```

## 6. Authentication

### 6.1 Overview

The security mechanics between Sign Live! cloud suite gears fsm backend and frontend are completely handled by Spring security (this chapter). This provides a broad range of features and protocols for authentication and application security, all implemented by an industry proven framework.

If you want to dive into configuring the security on your own, the Spring documentation (<https://docs.spring.io/spring-security/site/docs/5.3.2.RELEASE/reference/html5/>) is considered required prerequisite knowledge!

### 6.2 Opt-out

All chapters below describe in detail the security default configuration and some customizing options if you are OK with the overall design.

All decisions of the default security configuration are “opinionated” and provided as we believe this is a sound basic production configuration. For your reference, the default security configuration (“spring-gears-fsm-security.xml”) is provided in “example/spring-beans/gears fsm security default”.

If this design does not fit your needs, you can either customize (by reconfiguring spring beans) or completely opt out of the gears default security by adding the profile

```
spring.profiles.active=customSecurity
```

After this, Sign Live! cloud suite gears fsm backs off and spring security is off – spring boot auto configuration is switched off, too.

Now you can (and should) construct a spring security configuration of your own.

You can take the built-in default security configuration as an example.

### 6.3 Security Overview

Technically the spring security is implemented as a distinct Spring “SecurityFilterChain”.

While this is provided by a named bean, you should really keep away if you do not know what you are doing.

All secured endpoints like e.g. "start monitor" are using the path prefixes

- /api/v1/\*\*

The services require the role "OPERATOR" to be available.

The bean name is "securityRealmMonitor".

## 6.4 Logout

To ensure that no authentication is cached on the server side, you can leverage the logout endpoint. If there is any user in an associated session, the authentication information is discarded.

The endpoint is available at the path

```
/logout
```

Please note that this has no effect on data stored by your browser. If you enter authentication data in the dialog offered by the browser, the behavior is totally up to the browser. It may (or not) cache this information and you may (or not) be able to clear this information in a more or less simple way.

If you want to play around with the services using a plain browser, you may want to activate incognito mode upfront.

## 6.5 Authentication filter

The task of an authentication filter is to extract protocol specific information and inject it into the Spring security framework APIs.

With Spring you have a variety of authentication filters available. Details regarding the Spring security framework you can find here:

**<https://docs.spring.io/spring-security/site/docs/5.3.x/reference/html5/>**

If you decide to activate your custom spring security, the authentication filter needs to be replaced with the one required for your concrete security design. The id for the authentication filter bean that needs to be available is:

- securityRealmMonitorAuthenticationFilter

### 6.5.1 Static authentication

The first filter we want to introduce is provided by Sign Live! cloud suite gears fsm. As the security design requires an authenticated role to be present and we don't want to send authentication information, we inject the required authentication token and the authorities (roles) statically to satisfy the security checks.

The filter definition below injects an authenticated token with role "OPERATOR" for every call. This is the **default filter definition** you don't need to add this yourself.

---

#### Spring XML fragment

---

```
<bean
  id="securityRealmMonitorAuthenticationFilter"
  class="de.intarsys.spring.security.StaticAuthenticationFilter">
  <property name="authorities" value="ROLE_OPERATOR" />
</bean>
```

---

If you are using this filter, **no** security checks are made on behalf of the gears application. You should verify that the execution environment has the ability to filter all invalid requests (e.g. using VPNs, secured proxies, ...).

### 6.5.2 Basic authentication

Basic authentication is one of the most common authentication protocols used in the web (and its ok as long as you are using transport level security).

Basic auth requires a header

**Authorization: Basic <base64(user:password)>**

to be sent to the server. The server then extracts this information and looks up in an authentication manager (see below) if this combination is valid and what authorities are attached.

To activate this feature you can leverage the Spring framework **org.springframework.security.web.authentication.www.BasicAuthenticationFilter**.

---

#### Spring XML fragment

---

```
<bean id="securityRealmMonitorAuthenticationFilter"
  class="org.springframework.security.web.authentication.www.BasicAuthenticationFilter">
  <constructor-arg ref="securityRealmMonitorAuthenticationManager" />
</bean>
```

---

This definition injects a basic authentication filter. For authentication purposes the default authentication manager is used (more on authentication managers in the next section).

Now we have basic authentication in place, accepting only user/password combinations that are defined in the authentication manager.

### 6.5.3 OAuth2 authentication

Spring has broad support for OAuth2 token support.

An example can be found in `"/example/spring-beans/gears fsm security monitor oauth2/spring-gears-fsm-security-monitor-oauth2.xml"`. It can be

placed into e.g. the global config directory: “<cloudsuite.config.shared>/cloudsuite/config/modules”.

The Sign Live! cloud suite gears fsm UI also needs configuration. The necessary file can be found in “example/ui/ng/fsmConfiguration.json”. It can be placed into e.g. the global config directory: “<cloudsuite.config.shared>/cloudsuite/ config/ui/ng”.

## 6.6 Authentication manager

The task of an authentication manager is to take the provided IDs and credentials that have been extracted by the authentication filter and decide if they are valid and what authorities are associated (in our case, roles).

Again, Spring provides a framework and lots of default authentication providers to quickly get up and running.

Again an authentication manager needs to be replaced with the one required for your custom security design. The following authentication manager bean needs to be available:

- securityRealmMonitorAuthenticationManager

By **default**, it is mapped to an in-memory repository with the user, "operator" (password "operator").

---

### Spring XML fragment

---

```
<security:authentication-manager id="securityRealmMonitorAuthenticationManager">
  <security:authentication-provider>
    <security:user-service>
      <security:user name="operator" password="{noop}operator"
authorities="ROLE_OPERATOR" />
    </security:user-service>
  </security:authentication-provider>
</security:authentication-manager>
```

---

For sure you do not want to use this in production (except you have set up external authentication, but in this case, you should have a "StaticAuthenticationFilter" in place anyhow, effectively disabling authentication manager lookup).

### 6.6.1 In-memory repository

Let's assume you want to introduce another in-memory representation of your users. We enhance our example from the authentication filter chapter with an authentication manager.

## Spring XML fragment

```

<security:authentication-manager id="securityRealmMonitorAuthenticationManager">
  <security:authentication-provider>
    <security:user-service>
      <security:user name="foo" password="{noop}bar" authorities="ROLE_OPERATOR" />
    </security:user-service>
  </security:authentication-provider>
</security:authentication-manager>

<bean id="securityRealmMonitorAuthenticationFilter"
class="org.springframework.security.web.authentication.www.BasicAuthenticationFilter">
  <constructor-arg ref="securityRealmMonitorAuthenticationManager" />
</bean>

```

Now if you want to access the Sign Live! cloud suite gears fsm UI, you can enter with user "foo", password "bar".

You can find this configuration in the "/example/spring-beans/gears security monitor basicauth inmemory" folder of the Sign Live! cloud suite gears fsm product bundle.

## 6.7 Well known security beans

Just in case you feel tempted to tweak the built-in spring security definition, here are the well-known entry points. These beans can be overwritten in your own definition.

Keep in mind, that a security:http bean can not be overwritten or defined with an overlapping pattern. In this case, you have to add the profile "customSecurity" and rewrite a security definition from scratch.

Bean	Description
securityRealmMonitorAuthenticationManager	A spring <b>security:authentication-manager</b> definition.
securityRealmMonitorAuthenticationFilter	A servlet filter that provides the request authentication information.



## 7. Configuration

### 7.1 Overview

Sign Live! cloud suite gears fsm is highly configurable and comes with a bunch of possibilities where to tweak the installation. Here's an overview of the configuration mechanics.

The backend is based on Spring infrastructure and as such you have all well-known Spring customization tools at hand.

Whenever we reference an XML based configuration file, we use the term "bean definition", when we talk about Spring properties (key/value pair definitions) we use the term "property definition".

These terms are augmented with "built-in" when we mean hardcoded, pre-deployed definitions and "custom" when we talk of individual definitions invented by your configuration.

#### 7.1.1 Configuration location

The configuration tries to harmonize Windows and Linux based installations. We only use "abstract" location names by default. Here's the list of locations supported.

Variable	Description
cloudsuite.config.name	Name of configuration file to be used. Defaults to "gears-fsm".
cloudsuite.config.user	User individual configuration data. Here you can store e.g. individual property files. This location has highest precedence.
cloudsuite.config.shared	System wide configuration data. Here you can store e.g. system wide property files. This location overrides the built-in configuration and is overridden by the user individual configuration.
cloudsuite.data.shared	System wide state data. Here is where system specific databases, caches etc. will reside.
cloudsuite.temp.dir	Location for temporary data, defaults to java.io.tmpdir.
cloudsuite.log.dir	Location for writing log files.

These variables are mapped differently on different platforms to adhere to platform specific conventions.

All basic variables can be overwritten in the standard Spring way:

- Use a command line parameter to the Java VM  
-Dcloudsuite.data.shared=/srv/data/cloudsuite
- Use an environment variable  
CLOUDSUITE\_DATA\_SHARED=/srv/data/cloudsuite
- Use a key/value pair entry in a properties definition file like "gears.properties". It's understood that you cannot set the "cloudsuite.config.\*" properties in a property file (it would have only strange effects).

If a configured directory does not exist, the web application will try to create it. For this it will need write access to the parent directory. Alternatively, you can create the directories yourself and make sure the web application has permission to write to them.

### 7.1.1.1 Windows

Variable	Description
cloudsuite.config.name	gears-fsm
cloudsuite.config.user	%USERPROFILE%/cloudsuite/config
cloudsuite.config.shared	%ProgramData%/cloudsuite/config
cloudsuite.data.user	%USERPROFILE%/cloudsuite/data
cloudsuite.data.shared	%ProgramData%/cloudsuite/data
cloudsuite.temp.dir	%AppData%/local/temp
cloudsuite.log.dir	%ProgramData%/cloudsuite/log

### 7.1.1.2 Linux

Variable	Description
cloudsuite.config.name	gears-fsm
cloudsuite.config.user	<user home>/cloudsuite/config
cloudsuite.config.shared	/etc/cloudsuite
cloudsuite.data.user	<user home>/cloudsuite/data
cloudsuite.data.shared	/var/lib/cloudsuite
cloudsuite.temp.dir	/tmp
cloudsuite.log.dir	/var/log/cloudsuite

If you are using this default, you must create the directories and grant access to the user running the servlet container.

If you should deviate from this default, keep in mind that the values of "cloudsuite.config.user" and "cloudsuite.config.shared" must not point to the same target directory. The same applies for "cloudsuite.data.user" and "cloudsuite.data.shared".

Example: Linux hosting Tomcat servlet container

---

**Unix shell commands**


---

```

sudo mkdir /var/lib/cloudsuite
sudo chgrp tomcat /var/lib/cloudsuite
sudo chmod 770 /var/lib/cloudsuite

sudo mkdir /var/log/cloudsuite
sudo chgrp tomcat /var/log/cloudsuite
sudo chmod 770 /var/log/cloudsuite

```

---

## 7.1.2 Built-in configuration

Sign Live! cloud suite gears fsm is bootstrapped with built-in bean definitions that are contained in the WAR deployment.

There are two options to tweak this standard configuration.

First, some of the bean definitions (like the "dataSource", see below) are instrumented with Spring property definitions. This way you can fine-tune the bean using custom property definitions.

Second, bean definitions that are eligible for "overwriting" are always defined using a well-defined role name. You can create a bean definition with this role name in your custom bean definition to overwrite the system standard.

## 7.1.3 Custom property definition

By default, properties are defined with increasing priority from

- Built-in  
classpath:\${cloudsuite.config.name}.properties
- System level definition  
\${cloudsuite.config.shared}/\${cloudsuite.config.name}.properties
- User level definition  
\${cloudsuite.config.user}/\${cloudsuite.config.name}.properties

## 7.1.4 Custom bean definition

Custom bean definitions are read from the following locations, again with increasing priority:

- Built in  
classpath:spring-gears-fsm.xml  
classpath:modules/spring-gears-fsm-\*.xml
- System level  
\${cloudsuite.config.shared}/spring-gears-fsm-\*.xml  
\${cloudsuite.config.shared}/modules/spring-gears-fsm-\*.xml
- User level  
\${cloudsuite.config.user}/spring-gears-fsm-\*.xml  
\${cloudsuite.config.user}/modules/spring-gears-fsm-\*.xml

### 7.1.5 Using profiles

You can use Spring profiles to parameterize your configuration. A profile allows to bundle beans and properties and give them a meaningful name.

When starting the server, you can activate any of these profiles by defining

---

```
spring.profiles.active=<comma separated profile names>
```

---

This definition can be made in any of the well-known ways, either

- as environment variable
- as system property
- in your `${cloudsuite.config.name}.properties`

To restrict a bean definition to a specific profile, you can add a section in your configuration like this

---

#### Spring XML fragment

---

```
...
<beans profile="profilename">
...
</beans>
...
```

---

Any definition contained in the "<beans>" element will be used only if the corresponding profile is activated.

In addition, the application will read specific property files in addition to "`${cloudsuite.config.name}.properties`" whose name match "`${cloudsuite.config.name}-<profilename>.properties`".

The priority (increasing to the bottom) is

- System level definition
  - `${cloudsuite.config.shared}/${cloudsuite.config.name}.properties`
  - `${cloudsuite.config.shared}/${cloudsuite.config.name}-<profilename>.properties`,
  - In the order of profile definition, first one highest
- User level definition
  - `${cloudsuite.config.user}/${cloudsuite.config.name}.properties`
  - `${cloudsuite.config.user}/${cloudsuite.config.name}-<profilename>.properties`,
  - In the order of profile definition, first one highest

### 7.1.6 String expansion integration

The Sign Live! cloud suite gears fsm string expansion is integrated with the spring property definition to ease using custom properties at runtime.

You can use the prefix "config." for a spring property to make it visible in the "config" string expansion namespace.

For more information see the chapter "String expansion".

## 7.2 Web application root

In many production environments the web application container itself cannot know what the fully qualified external URL for the services is. This is most often caused by a reverse proxy.

While most of the time relative addresses are fine, sometimes the server needs to construct fully qualified URL, for example for redirect addresses.

While the server does its best to derive what may be the URL from an external point of view by examining de-facto standard HTTP headers, there may be times this is not working.

In such special cases you can override the automatic detection by setting the root URL explicitly using the property

---

```
de.intarsys.application.rootUrl
```

---

These are the locations where we look up the property from lowest to highest precedence.

You can set the property in the web.xml

---

**servlet container web.xml**

---

```
<context-param>
  <param-name>de.intarsys.application.rootUrl</param-name>
  <param-value>https://my.server.com</param-value>
</context-param>
```

---

You can use an environment variable

---

```
DE_INTARSYS_APPLICATION_ROOTURL=https://my.server.com
```

---

You can set the property using a Java system property

---

```
-Dde.intarsys.application.rootUrl=https://my.server.com
```

---

## 7.3 Logging

With regard to logging cloud suite tries to come up with a default setup that can be used out of the box.

Internally, the Logback logging framework is used. The features and syntax of Logback are beyond the scope of this documentation. There are many resources available on the internet.

### 7.3.1 Override logging

When starting up, Logback is configured using the default "logback.xml", situated **anywhere** in a root package on the class path (or known from the

Logback command line options). All standard Logback functionality is available. If you are happy with this and provide a configuration, you can skip the rest. As soon as cloud suite is aware of this "external" Logback configuration, it skips all further activities. You just must be **sure** that you do not have any of the cloud suite context variables at hand at this moment in time.

### 7.3.2 Built-in logging

If not overwritten, at the earliest moment in the application lifecycle (in a Spring listener), we perform a relaunch of the Logback environment - this time with the well-known cloud suite variables established.

We then try to load a "\${cloudsuite.config.user}/logback.xml" and "\${cloudsuite.config.shared}/logback.xml", if this fails, we fall back to an internal default Logback configuration.

This configuration has four appenders, STDOUT, ASYNCFILE, SIFT and SIFT\_CYCLIC.

RollingFileAppender:

- ASYNCFILE will write the whole application log to the "\${cloudsuite.log.dir}" directory - a platform dependent location as stated above, using the log level "INFO".
- SIFT will write log files for each monitor (SiftingAppender) – again to the "\${cloudsuite.log.dir}" directory.
- The encoding for the files is UTF-8.
- The total space usage is capped globally at 2GB. 1GB for all monitor logs combined and 1GB for the whole application log, meaning if the combined size exceeds this limit, older files will be removed.

Other appenders:

- STDOUT will write the whole application log to the console (ConsoleAppender).
- SIFT\_CYCLIC will keep the log of each monitor in memory (CyclicBufferAppender), for display and download in the Sign Live! cloud suite gears fsm UI.

The following Logback variables are available for your use within your logback.xml at this stage.

Logback variable	Definition
cloudsuite.config.shared	\${cloudsuite.config.shared}
cloudsuite.config.user	\${cloudsuite.config.user}
cloudsuite.data.shared	\${cloudsuite.data.shared}
cloudsuite.log.dir	\${cloudsuite.log.dir}
cloudsuite.log.level	\${cloudsuite.log.level}:INFO
config.shared	\${cloudsuite.config.shared}
config.user	\${cloudsuite.config.user}

data.shared	\${cloudsuite.data.shared}
log.dir	\${cloudsuite.log.dir}
log.level	\${cloudsuite.log.level}:INFO
log.name	\${cloudsuite.log.name}
log.id	LocalDateTime.now()

### 7.3.3 Log tweaks

If you don't want to provide a complete logback.xml configuration yourself, you can use one of the built-in configurations:

- <default>
- console

To select a configuration, you can set a property like this:

```
cloudsuite.log.config=console
```

#### 7.3.3.1 Default built-in configuration

The default configuration registers RollingFileAppenders, a ConsoleAppender and a CyclicBufferAppender. To activate it, just omit the corresponding property. You can try to use the following configuration hot-spots that are built-in into it.

#### 7.3.3.2 Directory

If you simply want to set another target directory, you can just set a property like this. This will be forwarded to the built-in log definition.

```
cloudsuite.log.directory=/srv/logs
```

#### 7.3.3.3 Level

If you simply want to set another logging level, you can just set a property like this. This will be forwarded to the built-in log definition.

```
cloudsuite.log.level=DEBUG
```

#### 7.3.3.4 Built-in configuration 'console'

The built-in configuration 'console' only enables console output for messages. This is particularly suitable for environments where file access is not available and logs are aggregated from the console output, e.g. on Cloud Foundry. You can activate it by setting the property like this:

```
cloudsuite.log.config=console
```

## 7.4 Licenses

The product requires valid licenses for execution.

Licenses are obtained from intarsys, a limited demo license is included with the product for basic usage.

You can provide new licenses by copying the license files either to

---

```
${cloudsuite.config.shared}/licenses
```

---

or

---

```
${cloudsuite.config.user}/licenses
```

---

After providing new licenses a restart is required.

Upon startup, all licenses that have been picked up are logged to the log file.

---

```
[04.05.2018-10:36:18.622][I][d.i.tools.license ][localhost-startStop-1][ ] loading
licenses from 'C:\ProgramData\cloudsuite\config'
[04.05.2018-10:36:18.637][D][d.i.s.d.pool.device ][rEnvironment service][ ] signature
pool launcher started
[04.05.2018-10:36:18.653][I][d.i.tools.license ][localhost-startStop-1][ ] license
'de.intarsys.cloudsuite.product.gears; 8; intarsys.de; 12/01/2017; 06/30/2018;
id=de.intarsys.cloudsuite.product.gears; account_automation_cli=-1:day;
account_automation_batch=-1:day; bundle=professional; batchsize=-1;
account_automation_api=-1:day; ' loaded
[04.05.2018-10:36:18.653][I][d.i.tools.license ][localhost-startStop-1][ ] license
'de.intarsys.security.device.common; 8; intarsys.de; 12/01/2017; 06/30/2018;
id=de.intarsys.security.device.common; de.intarsys.security.app.sign.account=-1:day;
de.intarsys.security.app.decrypt.account=-1:day; ' loaded
[04.05.2018-10:36:18.653][I][d.i.tools.license ][localhost-startStop-1][ ] license
'de.intarsys.security.device.bridge; 8; intarsys.de; 12/01/2017; 06/30/2018;
id=de.intarsys.security.device.bridge; de.intarsys.security.app.sign.account=-1:day;
de.intarsys.security.app.decrypt.account=-1:day; ' loaded
```

---



## 8. String expansion

### 8.1 Basics

#### 8.1.1 Why string expansion

Any non-trivial application needs variables to be adaptable to the usage context. Examples for this are

- configurable path to store temporary files
- host names to lookup information
- database URL's

String expansion allows for runtime replacement of dynamic content within strings.

Strings are used often in configuration and installation. Using string replacement, you gain access to environment information or runtime information, which gives you more flexibility during deployment and operation.

#### 8.1.2 Terminology

When talking about string expansion, we use two different concepts:

- Expression evaluation

“Expression evaluation” means replacing a variable name with its content, much like in any programming language you may know. If the variable **foo** has the value **bar**, then evaluating **foo** means replacing it with its value **bar**. Using this variable for example in a script, you can adapt it more easily to different customer needs by simply changing the variable.

- Template evaluation

While expression evaluation is quite useful by itself, it still can be improved. One problem you will encounter is how to differentiate a plain piece of text from a variable to be replaced, the other problem results from the need for more complex content that cannot be expressed using a single variable. Look for example at a directory name that should be built using the temporary directory plus the name of the current user.

These problems are addressed by the next level of evaluation - a template-based approach. A template is first a plain string. “Hello” is simply “Hello”. But a template is evaluated and scanned for special escapes, marking the

embedding of an expression. If it encounters such an expression, it is evaluated as described above and inserted in the original template string.

This is a quite common scenario and we will use it here to build our powerful string expansion. Our escapes will be `${` for marking the beginning and `}` for the end.

So, let's expand "Hello, `${username}`". Supposed there is a variable *username* containing the value *Nick* we will get the string "Hello, Nick". It's that simple.

### 8.1.3 Syntax

Embedding a variable in a string is preceded by `${` and ends with `}`. Enclosed is the name of the variable to be expanded.

```
hello, ${foo}
```

The variable *foo* is embedded in the string.

If you need a `${` in the text itself, you simply enclose it in an expression itself

```
${${}} is the escape sequence
```

will evaluate to

```
${} is the escape sequence.
```

### 8.1.4 Constant text in expression

While it seems a bit strange, constant text within an expression does make sense. The reason are the formatting features mentioned in a later chapter. They reach from number or date formatting to conditional evaluation. This is where constants come into play - you can define constant text that may **not** be contained in the evaluated template.

```
hello, ${"world"}
```

will evaluate to

```
hello, world.
```

### 8.1.5 Namespaces

String expansion can resolve an extensive set of named values from various sources. The values are organized into hierarchical namespaces. Accordingly, a value name can have a prefix of one or more namespace

name separated by dots ".", which specifies the path to the value. For example, our information is organized using the prefixes

- **properties** for selecting among VM properties
- **environment** for selecting execution context information like working directories

and many more.

Example

```
foo.bar.gnu.gnat.var
```

This is a valid name for "var" in the namespace "foo.bar.gnu.gnat"

You will find a complete description of the namespaces available in the chapters below.

## 8.1.6 Spring integration

The concept of string expansion is used throughout the SignLive! product family. With gears fsm we are facing the problem, that Spring uses the same escapes with their own string expansion implementation.

A problem arises when we want to use a template at runtime and configure it in a Spring XML or property file. Consider we want "signme.signer.username" to be replaced with the principal name at runtime. This example won't work:

```
signme.signer.username=${principal.user.name}
```

Spring will try to expand the value upon startup and fail. To be able to forward string expansion to the runtime expansion, we have to escape the escape...

One solution consists in reverting to the Spring expression language – resulting in an unreadable and error prone construct like this:

```
signme.signer.username=#{'$' + '{principal.user.name}'}
```

To support a more readable alternative, the application will post process Spring template processing by replacing all occurrences of "{?" with "\${". This way you can (and should) write:

```
signme.signer.username=${principal.user.name}
```

whenever you need your string expanded at runtime, not startup time.

Attention:

Replacing "{?" is done in spring configuration files only! Do not use this workaround in service arguments and other places.

## 8.2 Namespaces

### 8.2.1 Overview

Here we will learn about the most important namespaces and their respective variables available in Sign Live! cloud suite gears fsm. All of these namespaces are available in all expansion contexts in the application.

More information on special situations where you will have more information at hand will be found in the next chapter.

### 8.2.2 app

#### 8.2.2.1 Name

app

#### 8.2.2.2 Description

Access to information about the application.

#### 8.2.2.3 Variables

Name	Description
<b>name</b>	The application's name.
<b>version</b>	The full version of the application.
<b>major</b>	The major version of the application (may be empty).
<b>minor</b>	The minor version of the application (may be empty).
<b>micro</b>	The micro version of the application (may be empty).

#### 8.2.2.4 Availability

This namespace is available after application startup.

### 8.2.3 config

#### 8.2.3.1 Name

config

#### 8.2.3.2 Description

Access a Spring configuration subset.

#### 8.2.3.3 Variables

Name	Description
------	-------------

---

**\*** Any Spring property starting with "config."

---

#### 8.2.3.4 Availability

This namespace is available after application startup.

#### 8.2.3.5 Example

Given an entry in the gears.properties

```
config.foo=bar
```

this expression

```
example ${config.foo}
```

will evaluate to

```
example bar
```

### 8.2.4 counters

#### 8.2.4.1 Name

counters

#### 8.2.4.2 Description

Zero-based all-purpose counters.

#### 8.2.4.3 Variables

Name	Description
<b>&lt;name&gt;</b>	An arbitrarily named counter, for example, for creating unique ids. On the first request, the counter is initialized and returns 0. Each consecutive request increments the counter and returns the new value.

#### 8.2.4.4 Availability

This namespace is generally available.

### 8.2.5 entity

#### 8.2.5.1 Name

entity

## 8.2.5.2 Description

Characters that are difficult to type or to use in some contexts.

## 8.2.5.3 Variables

Name	Description
<b>amp</b>	ampersand &
<b>backslash</b>	backslash \
<b>copy</b>	copyright © (Unicode U+00A9)
<b>cr</b>	carriage return (Unicode U+000D, \r in Java strings)
<b>gt</b>	greater than >
<b>lf</b>	line feed (Unicode U+000A, \n in Java strings)
<b>lt</b>	less than <
<b>nl</b>	line separator of the VM (usually \n on Unix and \r\n on Windows)
<b>quot</b>	double quote "
<b>squot</b>	single quote '
<b>slash</b>	slash /
<b>trade</b>	trademark ® (Unicode U+2122)

## 8.2.5.4 Availability

This namespace is always available.

## 8.2.6 environment

## 8.2.6.1 Name

environment

## 8.2.6.2 Description

Access to the application's file environment. All paths are absolute.

## 8.2.6.3 Variables

Name	Description
<b>basedir</b>	The application's base directory. Most operations will be performed relative to this directory.

<b>profiledir</b>	The directory for shared configuration files \${cloudsuite.config.shared}
<b>datadir</b>	The directory for application-private data. \${cloudsuite.data.shared}
<b>workingdir</b>	The application's working directory.
<b>tempdir</b>	The application's directory for temporary files.

#### 8.2.6.4 Availability

This namespace is available after application startup for trusted use cases.

### 8.2.7 system

#### 8.2.7.1 Name

system

#### 8.2.7.2 Description

Access some system information.

#### 8.2.7.3 Variables

Name	Description
<b>architecture</b>	Returns the architecture of the current platform, which is either "64-bit" or "32-bit".
<b>getenv.&lt;name&gt;</b>	The value of the environment variable <name>.
<b>properties.&lt;name&gt;</b>	The value of the VM's system property <name>.

#### 8.2.7.4 Availability

This namespace is only available for trusted use cases.

#### 8.2.7.5 Example

```
example ${system.counter}
```

will evaluate to 0 when used for the first time and be incremented afterwards

```
→ example 32
```

```
example ${system.counters.test}
```

will evaluate to 0 when used for the first time and be incremented afterwards

```
→ example 0
```

## 8.2.8 time

### 8.2.8.1 Name

time

### 8.2.8.2 Description

Time-related properties.

### 8.2.8.3 Variables

Name	Description
<b>millis</b>	The difference, measured in milliseconds, between the current time and midnight, January 1, 1970 UTC.
<b>uniquemillis</b>	The difference, measured in milliseconds, between the current time and midnight, January 1, 1970 UTC. If multiple values are requested within the same millisecond, the result is delayed so that each value is unique.

### 8.2.8.4 Availability

This namespace is generally available.

## 8.2.9 file

### 8.2.9.1 Name

file

### 8.2.9.2 Description

File-related properties.

### 8.2.9.3 Variables

Name	Description
<b>name</b>	The name of a file (e.g. document.pdf)
<b>basename</b>	The name of a file without the extension (e.g. document)
<b>extension</b>	The file extension (e.g. .pdf)



#### 8.2.9.4 Availability

This namespace is available during the processing of a file.

### 8.2.10 event

#### 8.2.10.1 Name

event

#### 8.2.10.2 Description

Event-related properties.

#### 8.2.10.3 Variables

Name	Description
<b>params</b>	Parameters that relate to the file that is being processed
<b>attributes</b>	Attributes that are made available by the processing service

##### 8.2.10.3.1 params

Name	Description
<b>FILEPATH</b>	The absolute file path of a file that is being processed
<b>RELATIVEPATH</b>	The file path relative to the monitored directory

##### 8.2.10.3.2 attributes

Name	Description	Availability
<b>result</b>	Possible outcomes: <ul style="list-style-type: none"> <li>valid – when the validation result main indication is “TOTAL-PASSED”</li> <li>invalid – when the validation result main indication is “TOTAL-FAILED”</li> <li>unknown – when the validation result main indication is “INDETERMIANTE”</li> <li>noSignatures – when the validated document has no signatures</li> </ul>	Only while using the ValidatorService

<b>conformance</b>	Possible outcomes: <ul style="list-style-type: none"> <li>conformant – the targeted conformance level (or a higher level) was reached</li> <li>notConformant – the targeted conformance level was not reached</li> </ul>	Only while using the ValidatorService and service.args.request.conformanceLevel is set
<b>conformanceLevel</b>	Possible outcomes: <ul style="list-style-type: none"> <li>B</li> <li>V</li> <li>T</li> <li>LT</li> </ul> <p>For more information, please see the validator API documentation.</p>	Only while using the ValidatorService and service.args.request.conformanceLevel is set

#### 8.2.10.4 Availability

This namespace is available during the processing of a file found event.

## 8.3 Formatting

### 8.3.1 Overview

The content provided by the variables may sometimes be not well suited for use in a template. For example, take the `${system.millis}` which will expand to something like `162336576223`. If you use this, you most probably really want to see a formatted date, like `23.12.1987`.

In such cases you can post process the variable content using formatting instructions.

The formatting instruction is appended immediately to the variable expression, separated by a ":".

```
${foo:f}
```

or, in the case of hierarchical names

```
${foo.bar:f}
```

Formatting instructions process the result of the expression they are appended to. As such you can simply chain formatting instructions by simply adding more ":" separated instructions.

```
${foo.bar:*:dts}
```

This will recursively expand *foo.bar* (more to this later) and format the result as a date.

## 8.3.2 String formatting

### 8.3.2.1 Overview

String formatting is initiated by **s**. This conversion is applied by default.

This instruction allows you to convert the result of the evaluation to a string (if not already) and create suitable substrings.

If the result of the evaluation is not a String and no string conversion instruction is present, the default conversion is used.

### 8.3.2.2 Instruction

```
expr ":s" [ "(from, to)" ]
```

The evaluation result is converted to a string. The substring extending from *from* (inclusive) to *to* (inclusive) is used as the result of the formatting operation. If any of the values *from* or *to* is negative, the index is computed from the end of the string where *-1* is the last character in the string. The second parameter *to* may be omitted and is replaced to match the last character in the string.

### 8.3.2.3 Examples

With *variables.user.greeting* containing **hello, world**

```
example ${variables.user.greeting:s}
```

evaluates to

```
→ example hello, world
```

```
example ${variables.user.greeting:s(7)}
```

evaluates to

```
→ example world
```

```
example ${variables.user.greeting:s(0,4)}
```

evaluates to

```
→ example hello
```

### 8.3.3 Integer formatting

Integer number formatting is initiated by **i**.

This instruction allows you to convert number values to integer string representations.

#### 8.3.3.1 Instruction

```
expr ":i" [ "b" | "o" | "d" | "x" ]
```

The evaluation result is checked to be a number or convertible to a number. The integer part of the number is then returned as a string, written to the base defined as the second character in the instruction.

- **b** Binary representation
- **o** Octal representation
- **d** Decimal representation
- **x** Hexadecimal representation

#### 8.3.3.2 Examples

With *system.counter* containing '17'

```
example ${system.counter:i}
```

evaluates to

```
→ example 17
```

```
example ${system.counter:ib}
```

evaluates to

```
→ example 10001
```

```
example ${system.counter:io}
```

evaluates to

---

→ example 21

---



---

```
example ${system.counter:id}
```

---

evaluates to

---

→ example 17

---



---

```
example ${system.counter:ix}
```

---

evaluates to

---

→ example 11

---

### 8.3.4 Date formatting

Date formatting is initiated by **d**.

This instruction allows you to convert date values to “human readable” strings.

#### 8.3.4.1 Instruction

There are two flavors of date formatting, one using instruction characters that quickly reference a predefined format and the other using “pattern” strings that exactly describe the desired output format.

The evaluation result must be a date or a number. This date will be formatted. If the evaluation result is already a string, this string is returned without processing. Any other object will return an empty string.

---

```
expr ":d" [ "d" | "t" ] [ "s" | "m" | "F" ]
```

---

This first syntax creates output based on a predefined format. This formatting will always use the platform locale.

The optional first instruction character defines which part of the date will be used for formatting

- **no character** Date and time portion will be processed
- **d** Only the date portion will be used
- **t** Only the time portion will be used

The optional second instruction character defines the output format

- **no character** The full formatting is applied
- **s** Short formatting is applied
- **m** Medium formatting is applied

- **f** Full formatting is applied

With no formatting character available at all, a default formatting pattern is used suitable for a technical representation of a timestamp in a file name.

```
expr ":d(" pattern ")"
```

This second syntax creates output based on the format defined in the pattern. The pattern syntax is exactly as described by the standard Java runtime library. This formatting will always use the platform locale.

#### 8.3.4.2 Examples

With 'system.millis' containing a timestamp for the 12th of October, 2009 at 23 :33:11 and 1 milliseconds.

```
example ${system.millis:d}
```

evaluates to

```
→ example 2009_10_12-23_33_11_001
```

```
example ${system.millis:ddm}
```

evaluates to

```
→ example 12.10.2009
```

```
example ${system.millis:dts}
```

evaluates to

```
→ example 23:33:11
```

### 8.3.5 Float formatting

Float formatting is initiated by **f**.

This instruction allows you to convert numeric values to strings using a predefined pattern.

#### 8.3.5.1 Instruction

The evaluation result must be a number or convertible to a number. This number will be formatted.

```
expr ":f(" pattern ")"
```

This syntax creates output based on the format defined in the pattern. The pattern syntax is exactly as described by the standard Java runtime library. This formatting will always use the platform locale.

### 8.3.5.2 Examples

With *variables.user.price* containing **1234.567** and an US locale

```
example ${variables.user.price.f(0.0)}
```

evaluates to

```
→ example 1,234.6
```

```
example ${variables.user.price.f(000000)}
```

evaluates to

```
→ example 001235
```

## 8.3.6 File path formatting

File path formatting is initiated by **p**.

This instruction allows you to convert an evaluation result to a string that can be accepted by the underlying platform as a valid file name (including path separators). Every suspect character is simply replaced by an underscore **\_**.

### 8.3.6.1 Instruction

```
expr ":p"
```

The evaluation result is converted to a string, then every suspect character is replaced by an underscore.

### 8.3.6.2 Examples

With *variables.user.foo* containing **my\*?.file**

```
example ${variables.user.foo:p}
```

evaluates to

```
→ example my_.file
```

### 8.3.7 Default value

If the variable cannot be resolved, normally an exception is raised, either the current processing is terminated or your template will contain some text like "<expression evaluation failed>".

The default value instruction gives you control on what to do when evaluation fails.

#### 8.3.7.1 Instruction

```
expr "!"
```

Apply the expression after the "!" if the first one fails or evaluates to nothing.

#### 8.3.7.2 Example 1

Assuming that "foo" is undefined and "bar" holds "hello"

```
${foo:!bar} world
```

evaluates to

```
hello world
```

#### 8.3.7.3 Example 2

You can use literal expressions as default, too.

```
${foo:!'hello'} world
```

and

```
${foo:!"hello"} world
```

evaluate to

```
hello world
```

### 8.3.8 Recursion

The result of evaluating an expression may contain other variables - it needs to be reevaluated. For example, in this environment

- **variables.user.name** equals **Jim**
- **variables.global.greeting** equals **Hello, \${variables.user.name}**
- **variables.global.startmessage** equals **\${variables.global.greeting}!**  
**Your application is fully functional!**



The last expression should evaluate to “**Hello, Jim! Your application is fully functional!**”.

Simply applying the declarations as you see above will lead to **Hello, \${variables.user.name}! Your application is fully functional!** - The second iteration of replacement is missing. To add recursive re-evaluation, you must use this instruction.

#### 8.3.8.1 Instruction

```
expr ":*"
```

Recursively apply the string evaluation process to the result of evaluating this expression.

The nesting depth of recursive evaluations is restricted to 10. Remember that you can chain formatting instructions, for example to apply a string formatting first, followed by a deep evaluation.

#### 8.3.8.2 Example

So, the complete example for the above should be:

```
${variables.global.greeting:*}! Your application is fully functional!
```

evaluates to

```
Hello, Jim! Your application is fully functional!
```

### 8.3.9 Conditional evaluation

Sometimes a certain amount of decision is involved when expanding a template. A good example may be a template for a file to be moved by the system. If the file not already exists at the destination, you want it to have the same name as the original file. But, if a file with this name is already present you don't want it to be overwritten. Instead, you want the new file to get a new, unique name. You cannot create such a template for the filename with the features you have seen so far.

The solution is a “conditional” template. The result contains a certain part only if a condition associated with it is true. The host system evaluating the template injects the condition before evaluation.

#### 8.3.9.1 Instruction

```
expr "?:?" condition
```

The result of evaluating *expr* is inserted into the result value only if *condition* is **true**.

"condition" can be any expression that itself can be evaluated to "true", "t", "1" for → TRUE or "false"; "f", "0" for → FALSE.

To ease handling of conditions, "!" can be used to negate the condition result.

```
expr "?! " condition
```

### 8.3.9.2 Example

In the file system monitor scenario mentioned above, the system will evaluate the template for the output file name twice: The first time with the variable *collision* set to **false**. If the result of evaluation is not unique, the template is reevaluated, this time with *collision* set to **true**.

The above case for example may be written:

```
${path}/{system.millis:?collision}{"."?:?collision}${filename}
```

If evaluated with *collision* = *false* the result looks like *c:/temp/mydir/myfile.txt*. With *collision=true* it looks like *c:/temp/mydir/2983749287.myfile.txt*.

## 9. Appendices

### 9.1 Cheat sheet

#### 9.1.1 Windows locations

Variable	Description
cloudsuite.config.name	gears-fsm
cloudsuite.config.user	%USERPROFILE%/cloudsuite/config
cloudsuite.config.shared	%ProgramData%/cloudsuite/config
cloudsuite.data.user	%USERPROFILE%/cloudsuite/data
cloudsuite.data.shared	%ProgramData%/cloudsuite/data
cloudsuite.temp.dir	%AppData%/local/temp
cloudsuite.log.dir	%ProgramData%/cloudsuite/log

#### 9.1.2 Linux locations

Variable	Description
cloudsuite.config.name	gears-fsm
cloudsuite.config.user	<user home>/cloudsuite/config
cloudsuite.config.shared	/etc/cloudsuite
cloudsuite.data.user	<user home>/cloudsuite/data
cloudsuite.data.shared	/var/lib/cloudsuite
cloudsuite.temp.dir	/tmp
cloudsuite.log.dir	/var/log/cloudsuite

## 10. External References

---

[1] intarsys GmbH, Sign Live! cloud suite gears manual.

[2] intarsys GmbH, Sign Live! cloud suite validator manual.