ICT Documentation

Project Acronym:	**PEPPOL**
Grant Agreement number:	**224974**
Project Title:	**Pan-European Public Procurement Online**

PEPPOL eSignature Infrastructure Validation Client Components Documentation

Revision: 1.0

Authors:

Frank Schipplick (bremen online services)

Lars Thölken (bremen online services)

Project co-funded by the European Commission within the ICT Policy Support Programme		
Dissemination Level		
P	**Public**	**X**
C	**Confidential, only for members of the consortium and the Commission Services**	

Revision History

Revision	Date	Author	Organisation	Description
1.0	20100430	Lars Thölken	bos	First version

Statement of originality

This deliverable contains original unpublished work except where clearly indicated otherwise. Acknowledgement of previously published material and of the work of others has been made through appropriate citation, quotation or both.

Contributors

Organisations

bremen online services (main editor), Germany, http://www.bos-bremen.de

CNIPA (Centro Nazionale per l'Informatica nella Pubblica Amministrazione[1]) , Italy, www.cnipa.gov.it

DIFI (Direktoratet for forvaltning og IKT[2]) , Norway, www.difi.no

DILA (Direction de l'Administration Légale et Administrative Of French Prime Minister Office), France

InfoCamere, Italy, www.infocamere.it

Persons

Adriano Rossi, CNIPA

Ahmed Yacine, DILA

Alexander Funk, bos

Andreas Wall, bos

André Jens, bos

Daniel Eggert, bos

Edgar Thiel, bos

Frank Olthhoff, bos

Jon Olnes, DIFI

Lars Thölken, bos (editor)

Nils Büngener, bos

Piero Milani, InfoCamere

[1] From 29th December 2009, CNIPA will be renamed DigitPA (Legislative Decree 1st December 2009, n. 177)

[2] English: Agency for Public Management and eGovernment

Table of Contents

1 Introduction

1.1 Objective

This document provides the documentation of the client component “validation client”. To enable any PEPPOL pilot participants to use electronic signatures and certificate validation, WP1 provides a stand alone client component and a function library. These can handle signatures and the used certificates with respect to the PEPPOL WP1 specifications.

The software used for this is a modified version of a product of bremen online services, the “Governikus Signer”. It is shipped in two editions:

- **Governikus Signer Basic Edition**: Stand alone validation software, GUI based, “Signer” or “Signer basic” in the following and
- **Governikus Signer Integration Edition**: function library that provides functions for other software applications, “Signer Integration Edition” in the following.

These client components mentioned above are declared as foreground in terms of the PEPPOL project. This means, that they are not made available as open source and stay in the property right of the manufactuor, but they are nevertheless free to use during the project duration and afterwards.

1.2 Scope

This documentation describes a validation client software, which can be used to validate certificates or verify documents, utilising the PEPPOL validation infrastucture. The eSignature validation model can be used in many interoperability settings. In the PEPPOL context, it provides eSignature validation for exchange of business documents where eSignatures are required from the legal or the organisational layer in the European Interoperability Framework (EIF 2.0) interoperability model.

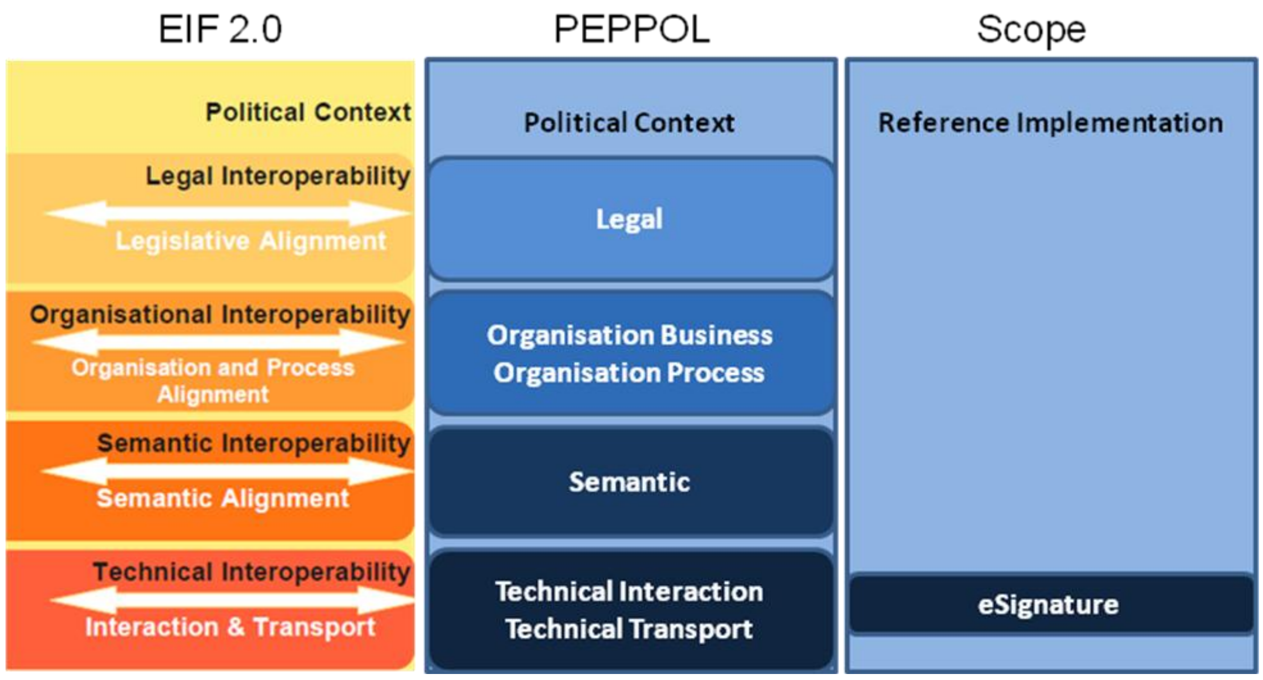


Figure 1: European Interoperability Layer model

Figure 1a shows which interoperability layers (EIF 2.0 and the PEPPOL specialization) that are in scope for this reference implementation. Figure 1b shows how the generic Reference Architecture for PEPPOL Interoperability (covering all interoperability layers) results in different Implementation models, that again sets the foundation fort he different Implementation Architectures and their (Reference) Implementations.

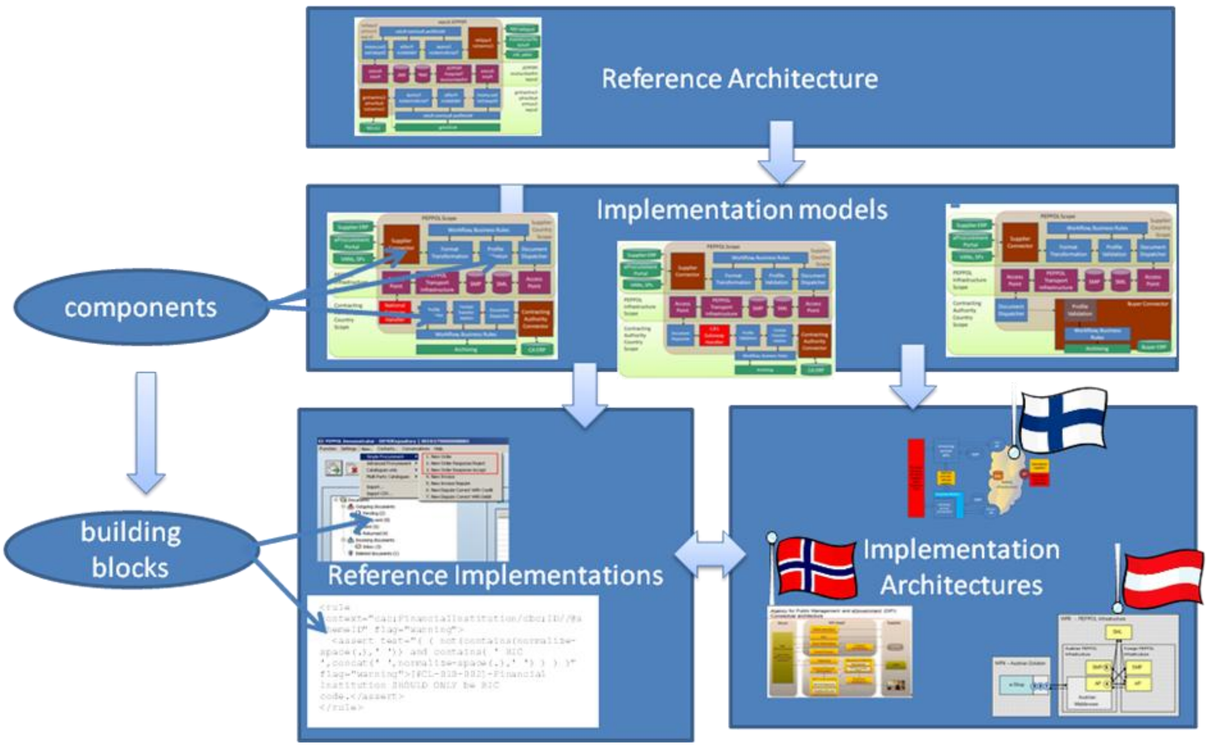


Figure 1b: PEPPOL Architecture-Implementation Model

1.3 Target Audience

Standard Users

The Documentation of the stand alone validation software basic edition is a user documentation that is designed to serve for software users with no special technical knowledge. It is assumed, that the readers have the ability to use standard software products.

Developers

The documentation of the validation client integration edition is designed to enable developers to integrate the functions of the Software into other applications. This ability is requesting a more profound knowledge in terms of software engineering.

1.4 Validation Client Version

This documentation refers to the version 2.2.2.0 of the Governikus Signer.

2 Validation Client Basic Edition

2.1 System requirements

Workstation

The following requirements must be met by your workstation:

- **Processor**: AMD or Intel
- **Memory**: Minimal 260 MB disk space, minimal 512 MB RAM
- **Monitor**: Minimal resolution1.024x768 pixels
- **Rights**: Administrative rights are required for installation and initialisation of the Governikus Signer/Integrated Signer. Further use does not require administrative rights.

Supported operating systems

The Governikus Signer can be run on the following operating systems.

- Windows XP
- Windows Vista
- Windows 7 (32 and 64 bit)
- Windows Server 2003
- Windows Server 2008
- openSUSE 11.2

Please mind that you have to call the respective installation files for Windows and Linux.

Note: All listed operating systems must be updated with their latest service packs. If not noted otherwise, only 32 bit systems are supported

Support for terminal servers

The following terminal server environments are supported.

- Citrix Metaframe Presentation Server 4.5
- Windows Terminal Server 2003
- Windows Terminal Server 2008

Detailed information to combinations of terminal servers, card readers and operating systems tested by the bos KG as well as notes for installation are given in the document "Unterstützte Kombinationen Terminalserver und Leser".

Java™ runtime environment

On the Governikus Signer installation a Java runtime environment (JRE) is installed as well, currently this is JRE version 1.6.0_11. This JRE will not interfere with any other Java installations and programs on your computer.

Requirements for electronic signatures

The following components are also needed:

- For creating advanced signatures an X509v3 certificate is needed (as software certificate or saved on a signature card)
- For creating a qualified signature a signature card of a listed certification authority is needed (see below).
- A card reader (not applicable if software certificates are used).

A list of all currently supported signature cards and card readers is included in delivery package.

Preconditions for using services

Certain Governikus Signer functions use server services:

- **Verification Server**: Online verification of qualified electronic signatures

You can use the Verification Server of the bos KG for online verifying qualified electronic signatures. This service is offered to all customers that use the Governikus Signer.

Here is the configuration data for the Verification Server of the bos KG:

- **Server name**: apollo.bremer-service.de
- **Port**: 80
- **Path**: Gov2Verificationserver/services/Gov2VerificationService

Note: If you connect to the Internet via a proxy server you must request the configuration data from your system administrator and set these in the Governikus Signer. Instructions are contained in the User Guide.

2.1.1 Legal information and further notices

Although this product documentation was written to the best of our knowledge and with reasonable care, errors and inaccuracies cannot be totally excluded. Legal or other liability for erroneous information and their consequences cannot be assumed. The information given in this product documentation reflects the current state of development and can be changed without further notice. Future editions can contain additional information. Technical und typographic errors are corrected in subsequent editions.

This product documentation as well as all copyright capable materials that are sold along with this product is copyright protected. All rights are reserved by the bremen online services Entwicklungs- und Betriebsgesellschaft mbH & Co. KG, Bremen, (bos KG). It is prohibited to copy or otherwise reproduce copyright capable materials without previous agreement. For legitimate users of this product this right is granted within the contractual terms. All copies of this product information or parts thereof must bear the same notice of copyrights, as does the original.

2.2 Introduction

The Governikus Signer offers all functionality necessary for securing and validating integrity and authenticity of files as well as for concealing file content. You can sign files with the Governikus Signer to ensure their integrity and authenticity. You can verify files to check their integrity and authenticity. You can even combine these processes.

If your workflow consists of mainly the same reoccurring settings you can mark these accordingly. You are then no longer prompted for these steps and the respective workflow steps are omitted, thus accelerating the Governikus Signer processes.

The legal and technical background of the Governikus Signer's functions is described in the chapter Explanations. In the chapters about the Governikus Signer's functions these explanations are hence omitted and dialogs and user guidance are explained. The user guide at hand is divided into the following main chapters:

- **Introduction**: The current chapter.
- **Installation**: Explains the steps for installing the Governikus Signer on your computer and references the hard- and software requirements for using the Governikus Signer. This is described in chapter 2.3.
- **Quick start**: Explains in short the basic steps for all functions. Within each step the according chapter with extensive explanations is referenced. This is described in chapter 2.4.
- **Settings**: Explains the central configuration of the Governikus Signer. You must only set these configurations once but you can always call this dialog again in case you want to change settings. This is described in chapter 2.5.
- **Main functions**: Introduces the description of the four main functions. At first it explains the dialog parts that are the same for all functions. This is described in chapter 2.6.

- **Sign**: Explains the steps for electronically signing one or more files. This is described in chapter 2.6.4.
- **Verify**: Explains the steps for verifying integrity and authenticity of an electronically signed file. This is described in chapter 2.6.5.
- **Explanations**: Explains vocabulary and backgrounds that are important in the Governikus Signer's context. This is described in chapter 2.7.
- **Secure display**: Explains the display program of the Governikus Signer that can visualise previously invisible changes to TIFF files and more. This is described in chapter 2.8.

2.3 Installation

Preconditions

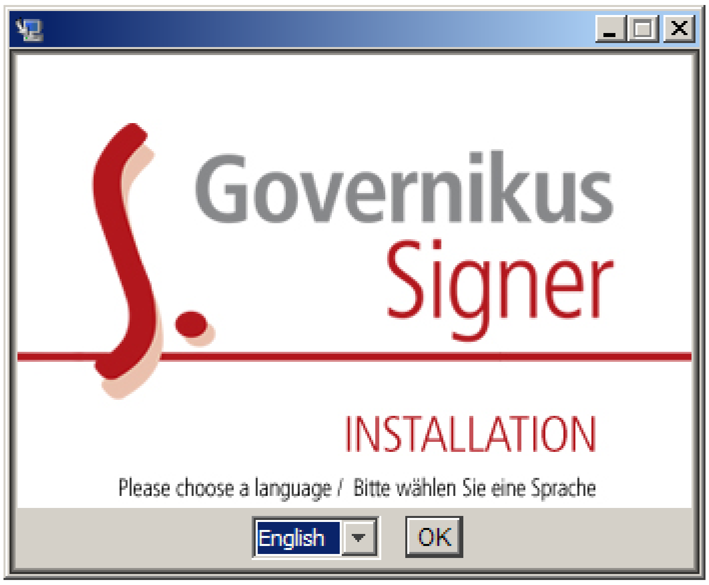
Please read the document "Governikus Signer system requirements" prior to installation. The preconditions for the installation are explained there. Furthermore, documents are available that inform you about which signature cards and card readers are supported. These documents are listed in the Governikus Signer's program group in the "start" - "All Programs" menu of your windows desktop and you can find these documents as well in your installation directory at:

```
<Governikus-Signer-installation-directory>\doc
```

Installation steps

The Governikus Signer is shipped as executable file. After calling the installation file you are lead step by step through the installation routine, where you can determine some settings. After the initial dialog you can always go back to the previous dialog by the back button, in case you want to change settings.

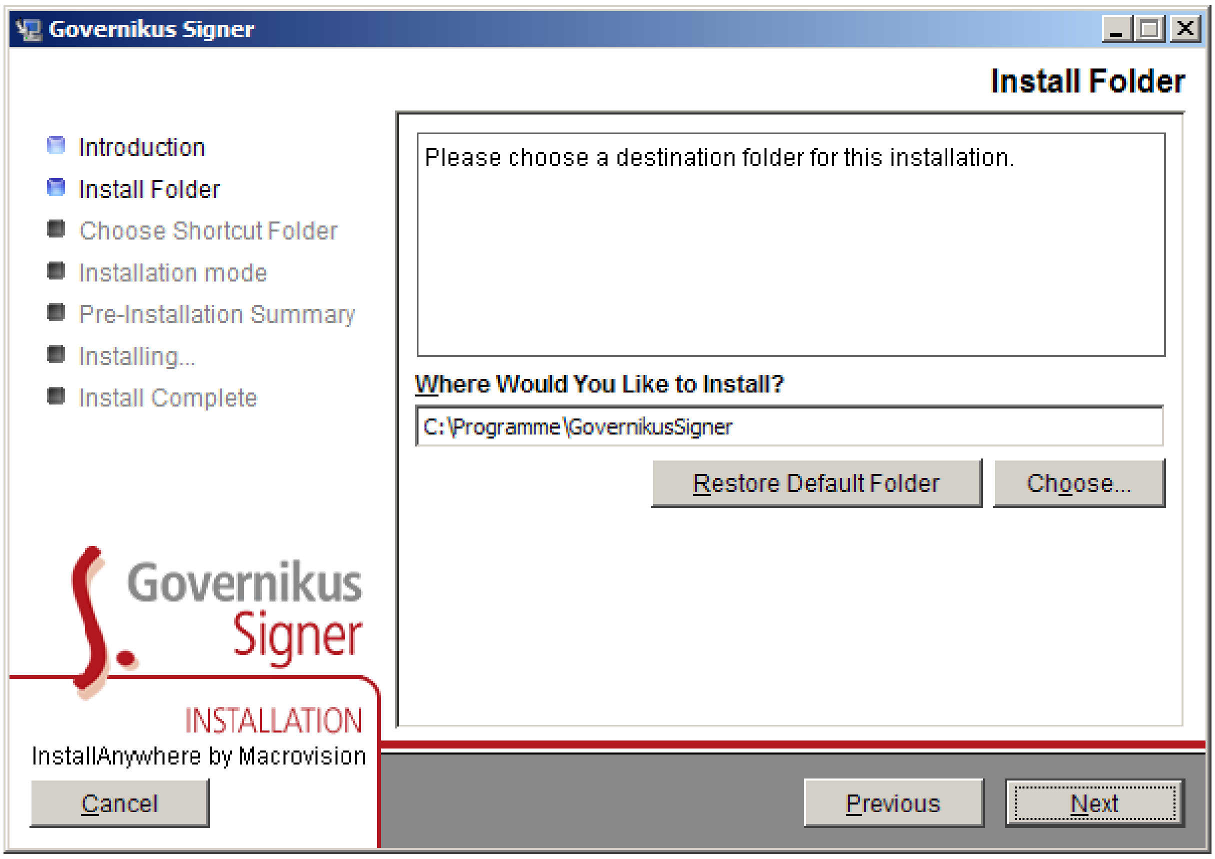
In the following dialog please select the language for the installation. The language selection has no effect on the user interface of the Governikus Signer.



Please choose a language / Bitte wählen Sie eine Sprache

Figure 2: Language selection in the first dialog window

In this dialog you can select an installation directory or you can accept the default.



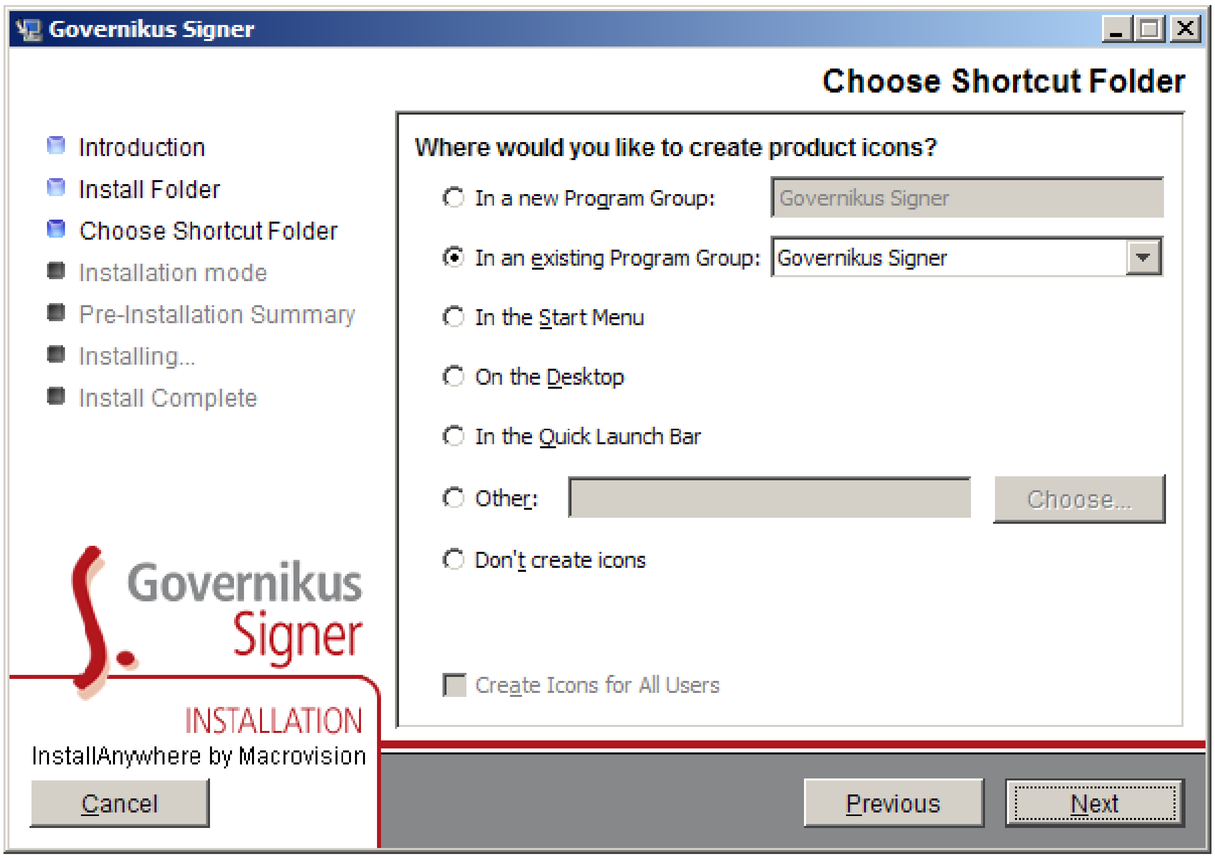
Please choose a destination folder for this installation.

Where Would You Like to Install?

C:\Programme\GovernikusSigner

Figure 3: Selection of the installation directory

Now you can select the associations for the program call.


Governikus Signer

Choose Shortcut Folder

- Introduction
- Install Folder
- Choose Shortcut Folder
- Installation mode
- Pre-Installation Summary
- Installing...
- Install Complete

Where would you like to create product icons?

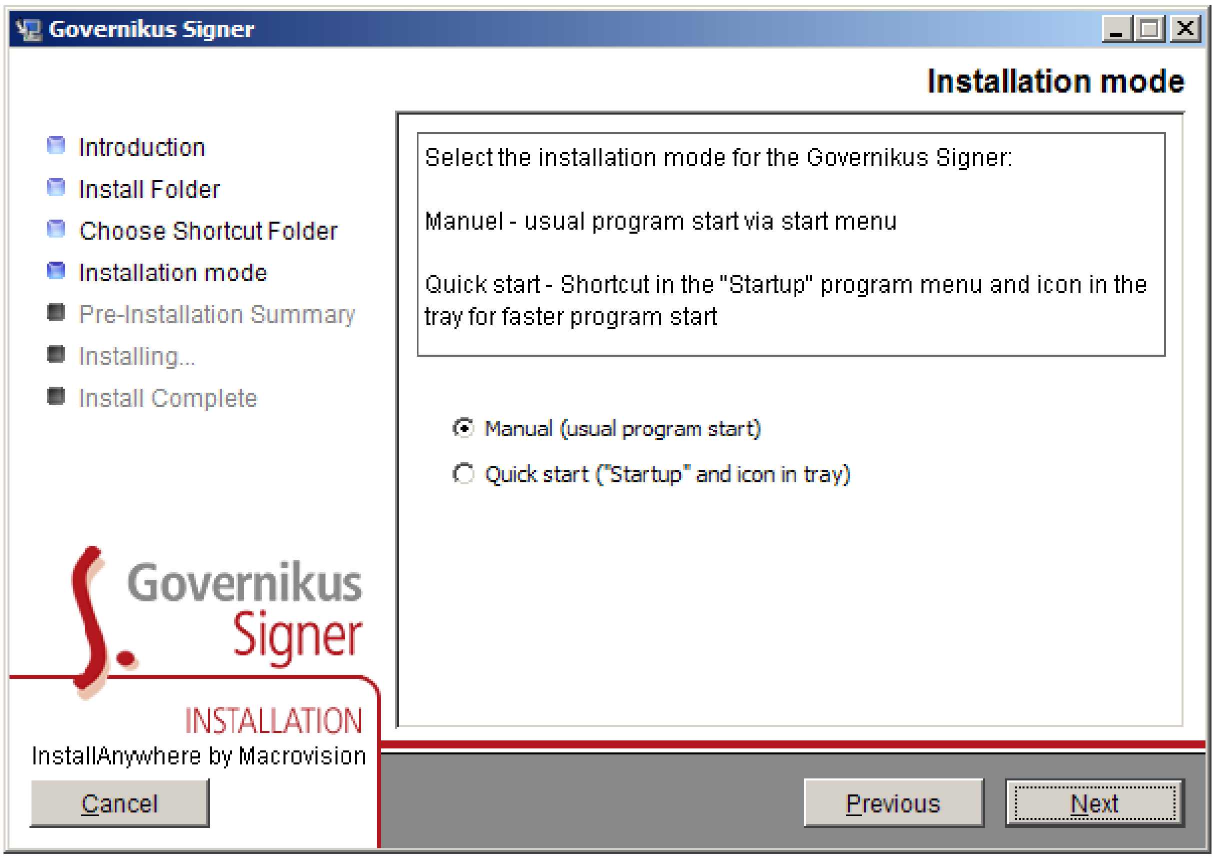
- ○ In a new Program Group: Governikus Signer
- [✓ In an existing Program Group: Governikus Signer]
- ○ In the Start Menu
- ○ On the Desktop
- ○ In the Quick Launch Bar
- ○ Other:
- ○ Don't create icons

☐ Create Icons for All Users

Figure 4: Associations for the program call

Choose the installation mode on the following dialog page. You have two options:

- Manual: With this installation mode only the selection is applied that you have chosen on the previous page. The program is started via an icon on the desktop (if selected) or via the start menu and the respective program selection.
- Quick start: In addition to your selection on the previous page this selection adds an entry to the "Startup" folder. In the system tray an icon is provided for quick starting the Governikus Signer. This prepares the Governikus Signer for a quick call of the user interface. The program is started in the background. Please mind that this installation mode delays booting your computer a little and permanently requires approximately 80 MB memory (RAM).



Installation mode

Select the installation mode for the Governikus Signer:

Manuel - usual program start via start menu

Quick start - Shortcut in the "Startup" program menu and icon in the tray for faster program start

[✓ Manual (usual program start)]
Quick start ("Startup" and icon in tray)

Figure 5: Selection of the installation mode

After this dialog your installation settings are summarised on the next page. Click on "Install". After installation click on "Done" to finish the installation. You can now call the Governikus Signer.

Note: For operating Governikus Signer on a Windows Server 2003 environment the MultiClient.exe must be started first. This program administrates the individual Governikus Signer instances for remote users.

MultiClient.exe

This program administrates the individual instances of Governikus Signer of the remote users (Windows Server 2003 only). It starts the web service servicePorts?wsdl on port 8086. In case this port is not available or already in use, another port can be determined with the environment variable "SERVICEPORT". However, you have to mind that every Governikus Signer instance needs two consecutive ports. Hence, the ports above the given service ports should be available.

Connecting card readers

Card readers that are connected via USB usually don't require further installation steps by users. Card readers that are connected at a serial port usually require separate drivers that must be manually installed if necessary. Please mind the documentation of your card reader in this case. If you are using a card reader on Linux the PCSC daemon must be started.

Note: For card readers on Linux system we recommend the newest software for CCID drivers and PCSC lite that you can download from this web page: http://pcsclite.alioth.debian.org

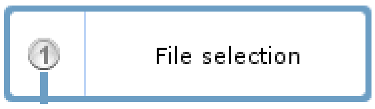
2.4 Getting quickly started

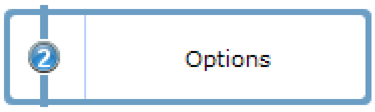
The following pages offer you a quick start for the Governikus Signer's functions. Using the Governikus Signer is simple and almost self-explanatory. All steps are explained in-depth in subsequent chapters that you can reach via the respective links.

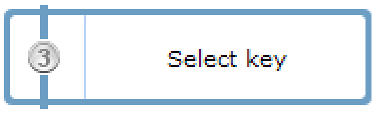
2.4.1 Quick start signing

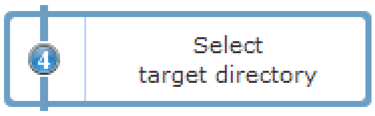
On the Governikus Signer's start-page click on "Sign", alternatively you can use the shortcut "Alt + s" or "Ctrl + F1".

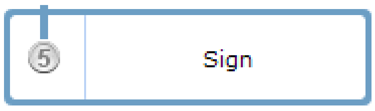
After calling the signing function a button bar is displayed that enables selecting different dialogs. Click the buttons one after the other to make your selections.

- 

Select the files that you want to sign. You can as well select the files for signing in the file manager and select "Sign" from the context menu. You can find detailed explanations in chapter 2.6.4.1.
- 

Select the signature format here. You can find detailed explanations in chapter 2.6.4.2.
- 

Select the certificate and key for your electronic signature. You can find detailed explanations in chapter 2.6.4.3.
- 

Select the target directory that will contain the electronically signed files. You can find detailed explanations in chapter 2.6.4.4.
- 

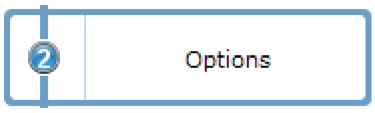
After your above selections, initiate the signing function here. You can find detailed explanations in chapter 2.6.4.5.

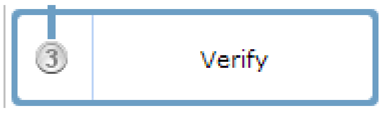
2.4.2 Quick start verifying

On the Governikus Signer's start-page click on "Verify", alternatively you can use the shortcut "Alt + v" or "Ctrl + F2".

After calling the verifying function a button bar is displayed that enables selecting different dialogs. Click the buttons one after the other to make your selections.

-

Select the files that you want to verify. You can as well select the files for verifying in the file manager and select "Verify" from the context menu. You can find detailed explanations in chapter 2.6.5.1.
- 

Decide here whether an online verification is wanted. Select a directory that will contain the inspection sheet of the verification. You can find detailed explanations in chapter 2.6.5.2.
- 

After your above selections, initiate the verifying function here. You can find detailed explanations in chapter 2.6.5.3.

2.5 Settings

Call this dialog by the "Settings" option in the "Extras" menu or use the shortcut "Ctrl + 1". The settings dialog contains several tabs that are explained in the following. You can find the following buttons on every tab in the settings dialog:

- Save: This button saves your settings. If you have altered settings on several tabs it is sufficient to use this button once on any of the tabs to save all your changes. You are leaving the settings dialog by using the save button.
- Cancel: You are leaving the settings dialog with this button. If you have made changes these are omitted. All changes to settings done after your last save on any of the tabs are lost.
- Magnifier: This option creates a red frame in which the selected section is displayed enlarged. Click into the frame to leave the magnifier function.
- Help: This buttons calls the online help of the Governikus Signer.

Common shortcuts in the settings dialog

- Ctrl + s = Save configuration and close settings window.
- Esc = Cancel
- F1 = Help

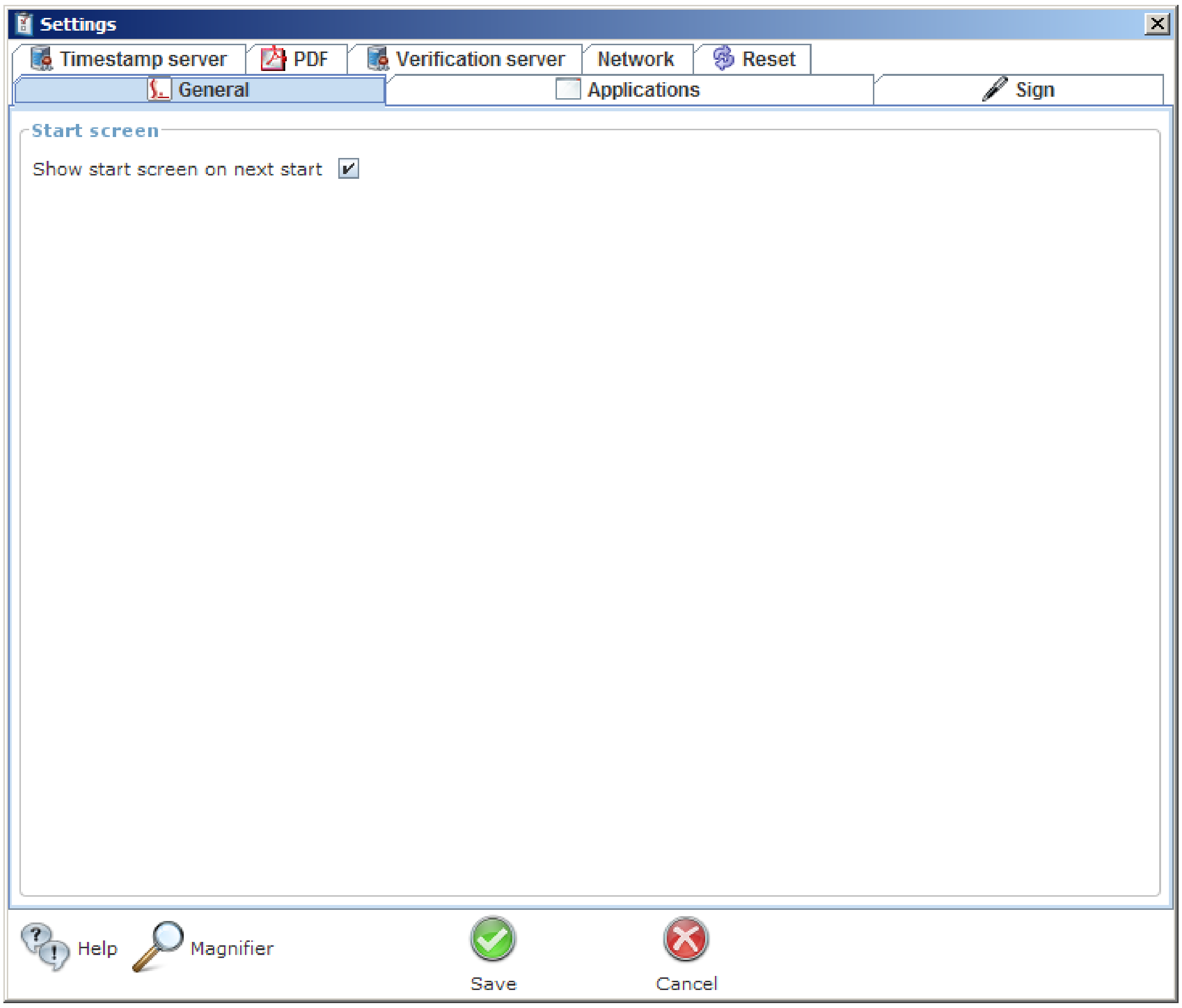
2.5.1 The general tab

This tab offers the following settings:

- **Start screen**: You can choose whether the start screen is shown on calling the Governikus Signer or not. However, we recommend displaying the start screen since Governikus Signer takes some time to start, which is indicated by the start screen. Without start screen you don't have any feedback during program start.

- **Logging**: If you select this option you have to choose a directory, which will contain the log files. Use the "Select logging file" button. These files contain the logging for errors and exceptions that may occur during program execution. Certainly, your PIN and other security relevant input is not logged. After selecting the logging directory, its path is displayed next to the input field. You can as well enter the path directly into the input field. In case the given directory does not exist, it is created. On every program call a log file is created in the given directory that has the name `<date-of-creation>_GovernikusSigner<version-number>.log`, wherein `<date-of-creation>` contains date and time of your Governikus Signer's program start and `<version-number>` denotes the version of your Governikus Signer.

The following figure shows the settings dialog with the "General" tab.



Start screen

Show start screen on next start ☑

Figure 6: "General" tab on the settings dialog

Shortcuts on this dialog

- Alt + l = select log file directory

2.5.2 The Applications tab

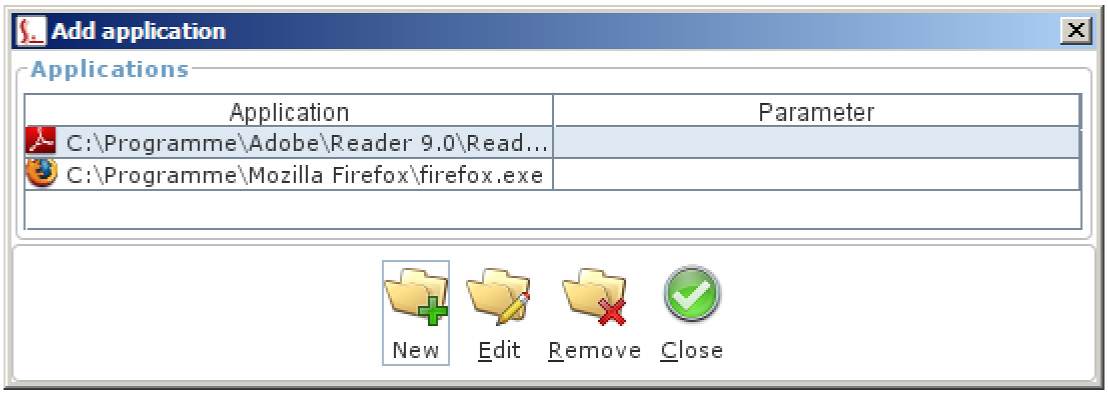
This tab offers configuring programs that are executed at the end of a Governikus Signer's function. On the last dialog of the Governikus Signer's functions "Sign" and "Verify" it is offered to choose a program from a list of programs, which you have configured here. The result files that are created on this last dialog are directly passed to the program selected and further processed with this program.

Example

If for example the files are signed on the last dialog of the Governikus Signer's "Sign" function and you have selected to commence with an e-mail program, then the signed files are passed to the configured e-mail program, each signed file to its own message.

Manage applications

Click on this button to add, edit or delete programs. The following dialog is shown:



Application	Parameter
C:\Programme\Adobe\Reader 9.0\Read...	
C:\Programme\Mozilla Firefox\firefox.exe	

Figure 7: Manage applications dialog

Shortcuts on the dialog "Add application"

- Alt + a = add application, Alt + e = edit, Alt + r = remove
- Del = remove
- Alt + c = Close
- Esc = close dialog

You can either edit the settings of a listed application or add a new program.

- For editing select a program from the list and click the edit button. The dialog is invoked and displays the current settings.
- If you click "New" the same dialog is invoked, however no settings are displayed.

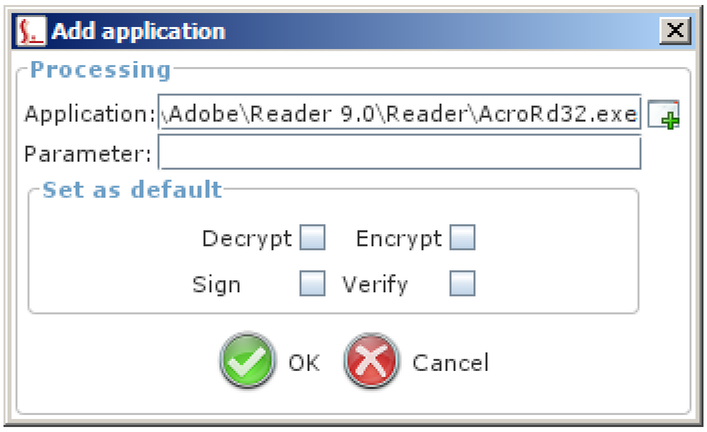


Figure 8: Edit or add program dialog

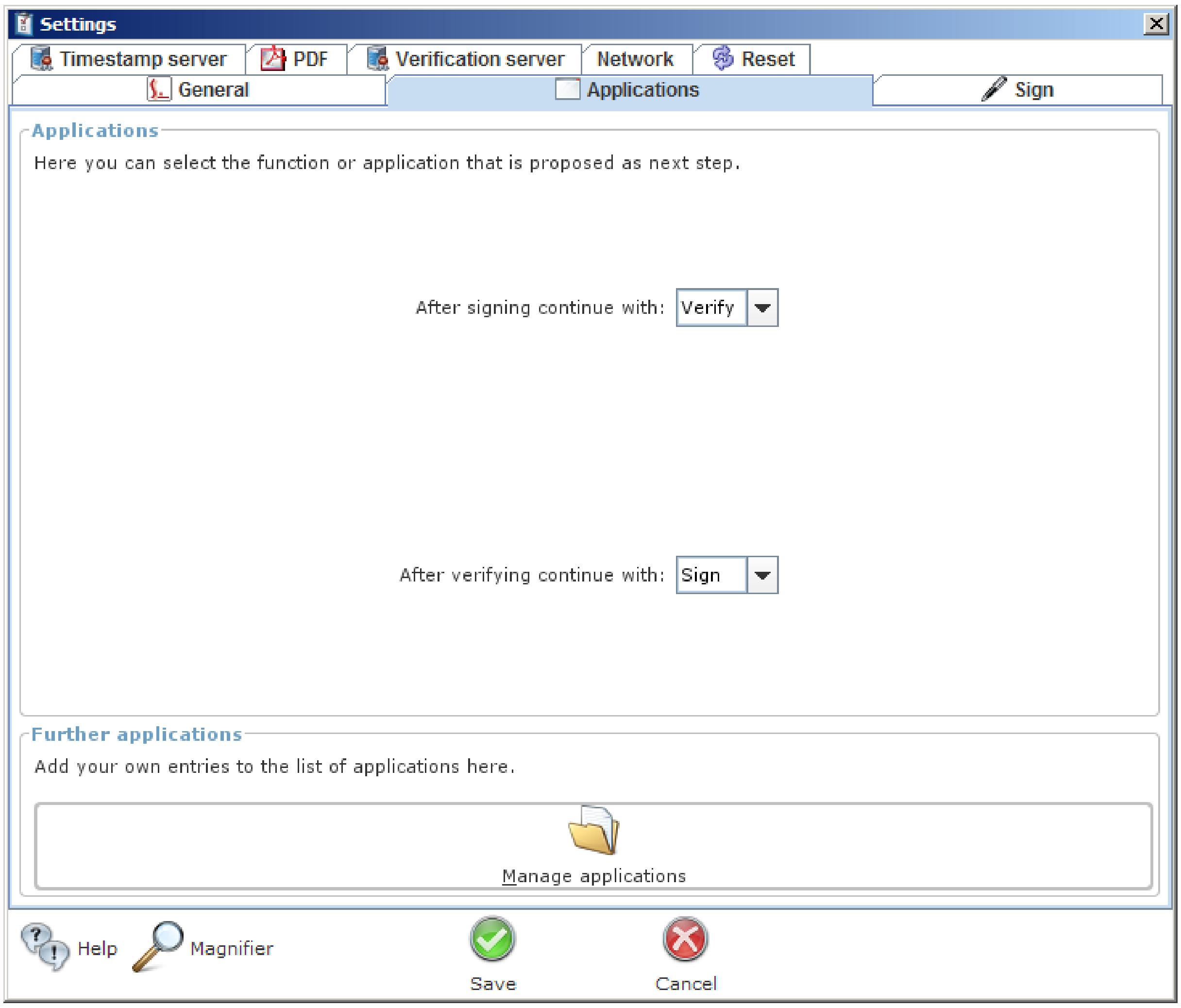
Use the green "plus" symbol on the right hand side to the row "Application" to navigate to the executable file of the program you want to add. Path and file are then displayed in the "Application" field. Add possibly required parameters in the "Parameter" field. Which parameters in which syntax are to be added depends on the selected application. If required ask your system administrator for these settings.

Shortcuts for the "Add application" dialog

- Alt + o = Opens the file chooser to navigate to the respective application
- Enter = OK
- Esc = Cancel

Processing tab overview

The following figure shows the "Processing" tab in the settings dialog.



Applications

Here you can select the function or application that is proposed as next step.

After signing continue with: Verify

After verifying continue with: Sign

Further applications

Add your own entries to the list of applications here.

Figure 9: "Applications" tab

Shortcuts on this dialog

- Alt + m = Invokes the "Add application" dialog

2.5.3 The Sign tab

Number of files to be viewed

According to the German Digital Signature Act §17 a software that enables electronic signing of files must offer the possibility to visualise the respective files' content prior to signing.

Viewing of files required: If you select this checkbox you can fill in a number here.

Minimum number: This number denotes a contingent of files in percent that must be viewed before the Governikus Signer can start signing electronically. Signing is only de-blocked after the respective amount of files was viewed. To be on the save side you should enter 100. In case this is not feasible for your workflow and you can trust the correctness of the file contingent in the file list, you can enter any number from 1 to 100.

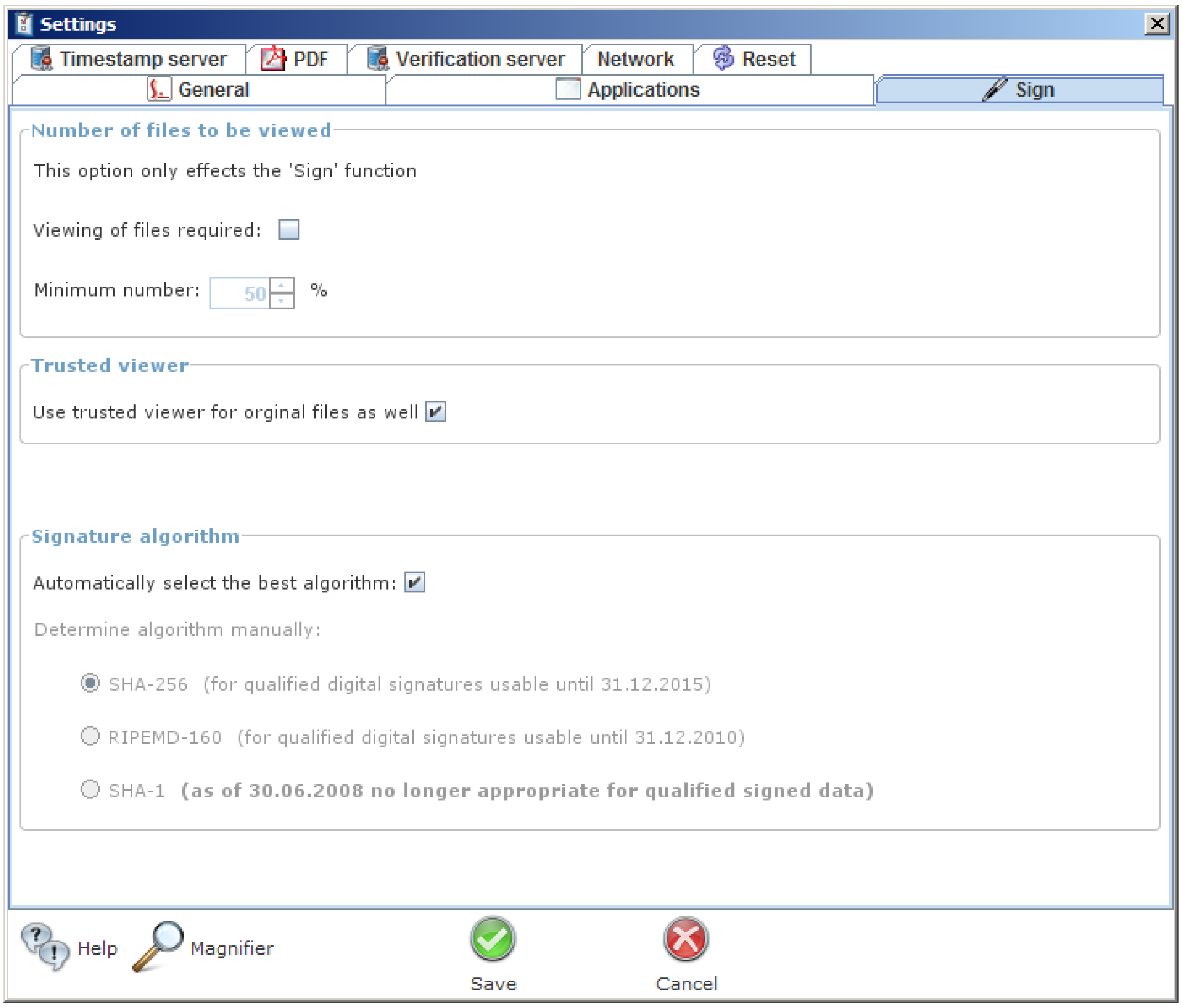
Trusted viewer

- **Use trusted viewer for original files as well**: Select this option if you want to use the trusted viewer for files in the file selection dialog (first dialog of the "Sign" function). If you deselect this option, the files are displayed by the program that is associated with the respective file type on your computer.

Signature algorithm

- **Automatically select the best algorithm**: Use this option to automatically set the best algorithm for signing.
- **Determine algorithm manually**: If you haven't selected the above checkbox you can set the algorithm here that is to be used for signing your files. The most secure algorithm is the first one.

The following figure shows the "Sing" tab in the settings dialog.



Number of files to be viewed

This option only effects the 'Sign' function

Viewing of files required: ☐

Minimum number: 50 %

Trusted viewer

Use trusted viewer for orginal files as well ☑

Signature algorithm

Automatically select the best algorithm: ☑

Determine algorithm manually:

● SHA-256 (for qualified digital signatures usable until 31.12.2015)

○ RIPEMD-160 (for qualified digital signatures usable until 31.12.2010)

○ SHA-1 **(as of 30.06.2008 no longer appropriate for qualified signed data)**

Figure 10: "Sign" tab

2.5.4 The PDF tab

These options only apply for settings of embedded signatures in PDF files (PDF inline). If you have activated the extended signature mode in the options of the "Sign" function (see chapter 2.6.4.2) you extend the signature with further information like cause of signature, location and date and add a visible image in the PDF file.

Additional signing information

You can add additional information to your signatures in this section. This information is placed within the document and can be visualised with an appropriate viewing program (e.g. Adobe Reader). It can be visualised and is printable from within the document.

Signee

Select here how your personal data is to be visualised:

- **Take from signature certificate**: If you select this radio button the name in the signature certificate is taken for each signature.

Note: The following options are only available if the drop-down-list "Type" in the subsequent dialog section contains a visualisation type. If "None" is selected the following options are greyed out.

- **Name**: If you chose this radio button, you can enter any name (e.g. a shortened Christian name) of at most 50 characters, or you can leave this field blank if you don't want to enter a signee.
- **Location**: You can enter a location of at most 50 characters or you can leave the field blank.
- **Date**: If you select the checkbox the current date is added. Please mind that the signing time is always contained within the signature. Hence, the date as additional information only makes sense in connection with the visualisation described in the following.
    - **Format**: Select the date's representation.

The cause for signing also belongs to the signing information. Since a frequent change can be expected here, the input of the signing cause is placed directly in the "Sign" dialog in the "Options" section.

Visualisation

In this section you can determine the scope, the display, and the position of the visible signature information. You can either add the extended signature information described above, an image or both.

You have to enter the entire size as rectangle and the proportions of size between text and image if you want to add text and image. The size of the selected image is automatically adjusted to the available space, while the proportions of the rectangle are kept. The examples in the following figure show these proportions. Details of the settings are described in the following section.

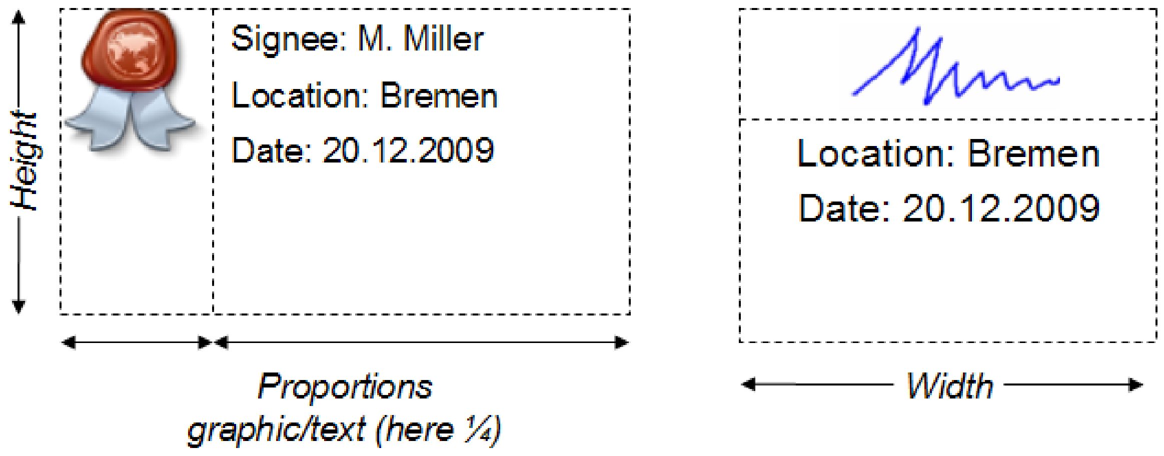


Figure 11: Examples for a visualisation with image and text

The dotted lines in the above figure are only inserted for exemplification. The dialog offers the following settings:

- **Type:** Here you can select whether a visible display is shown at all and whether it should display additional signee information, an image or both.
- **Layout:** Determine the order of text and image.
- **Page:** Determine whether the display is placed on the first or last page of the document.
- **Height:** Determine which height the visible signee information should have.
- **Width:** Determine the width of the visible signee information.
- **Image/text:** Determine the proportions of image and text. The entire size results from the values in the field height and the field width. If you have selected text and image side by side the section is horizontal, otherwise vertical.
- **Position:** Determine the position of the visible signee information on the document page. Hold the red frame with the mouse and drag it to the position where you want to display the information.

In the next dialog section you can set details of the text display as well as the image to be used.

Font

Here you can set details of the text representation.

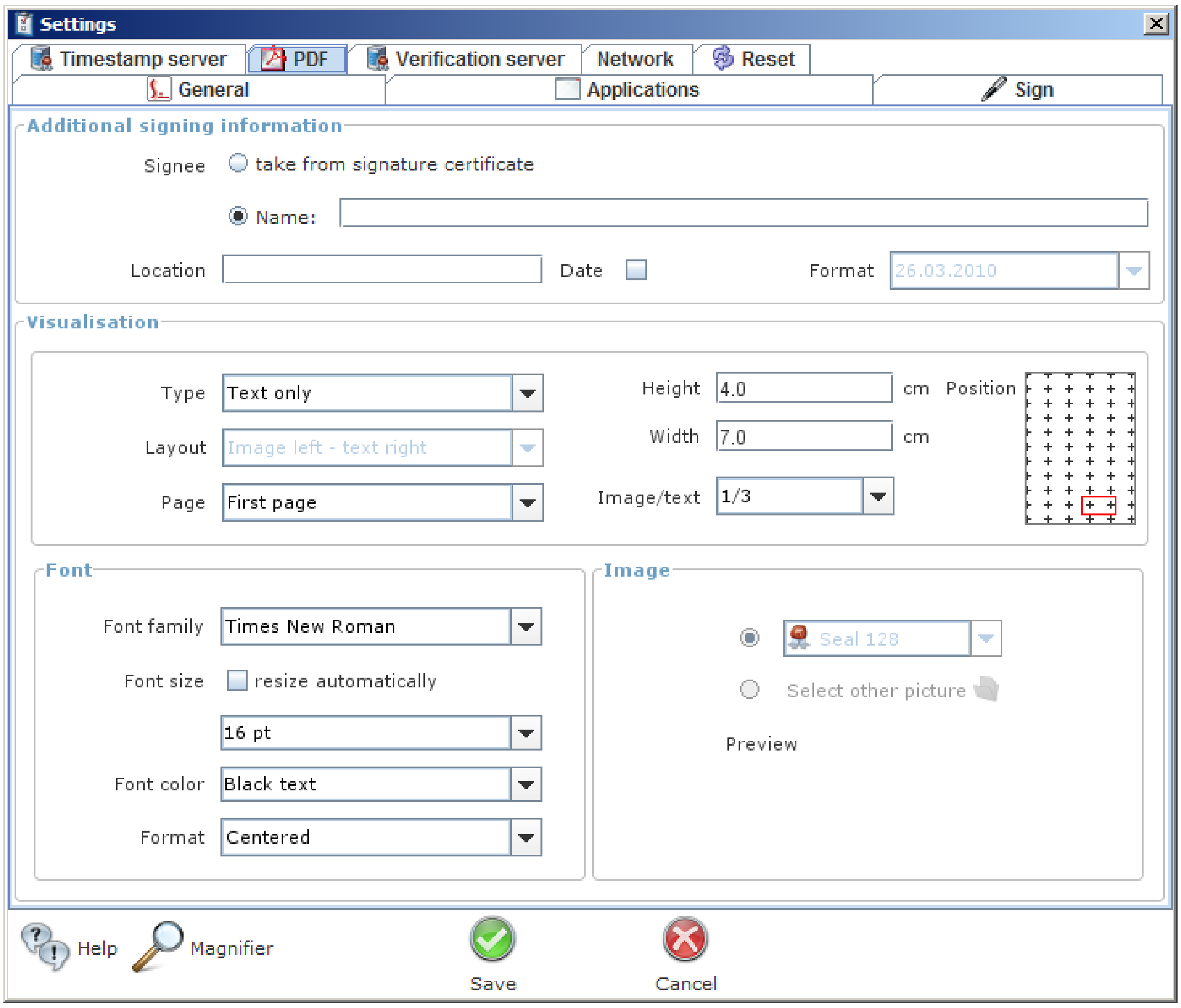
- **Font family**: Since displaying of the documents should be possible on various operation systems the font family selection is restricted to "Times New Roman", "Helvetica" and "Courier New".
- **Font size**: Select the font size from the drop-down-list.
  - **resize automatically**: If this checkbox is activated the font size is automatically reduced if the text doesn't fit the defined space.
- **Font colour**: Select the colour from the drop-down-list.
- **Format**: Select the alignment from the drop-down-list.

Image

Determine which image is to be inserted. You can either select an image from the drop-down-list or upload an image with a file chooser dialog. The image must have the format PNG, JPG or GIF. The selected image is shown in a preview.

PDF tab overview

The following figure shows the "PDF" tab in the settings dialog.



Additional signing information
Signee: take from signature certificate / [✓ Name:]
Location: (empty)
Date: ☐
Format: 26.03.2010

Visualisation
Type: Text only
Layout: Image left - text right
Page: First page
Height: 4.0 cm
Width: 7.0 cm
Image/text: 1/3
Position

Font
Font family: Times New Roman
Font size: ☐ resize automatically
16 pt
Font color: Black text
Format: Centered

Image
[✓ Seal 128]
Select other picture
Preview

Figure 12: "PDF tab" in the settings dialog

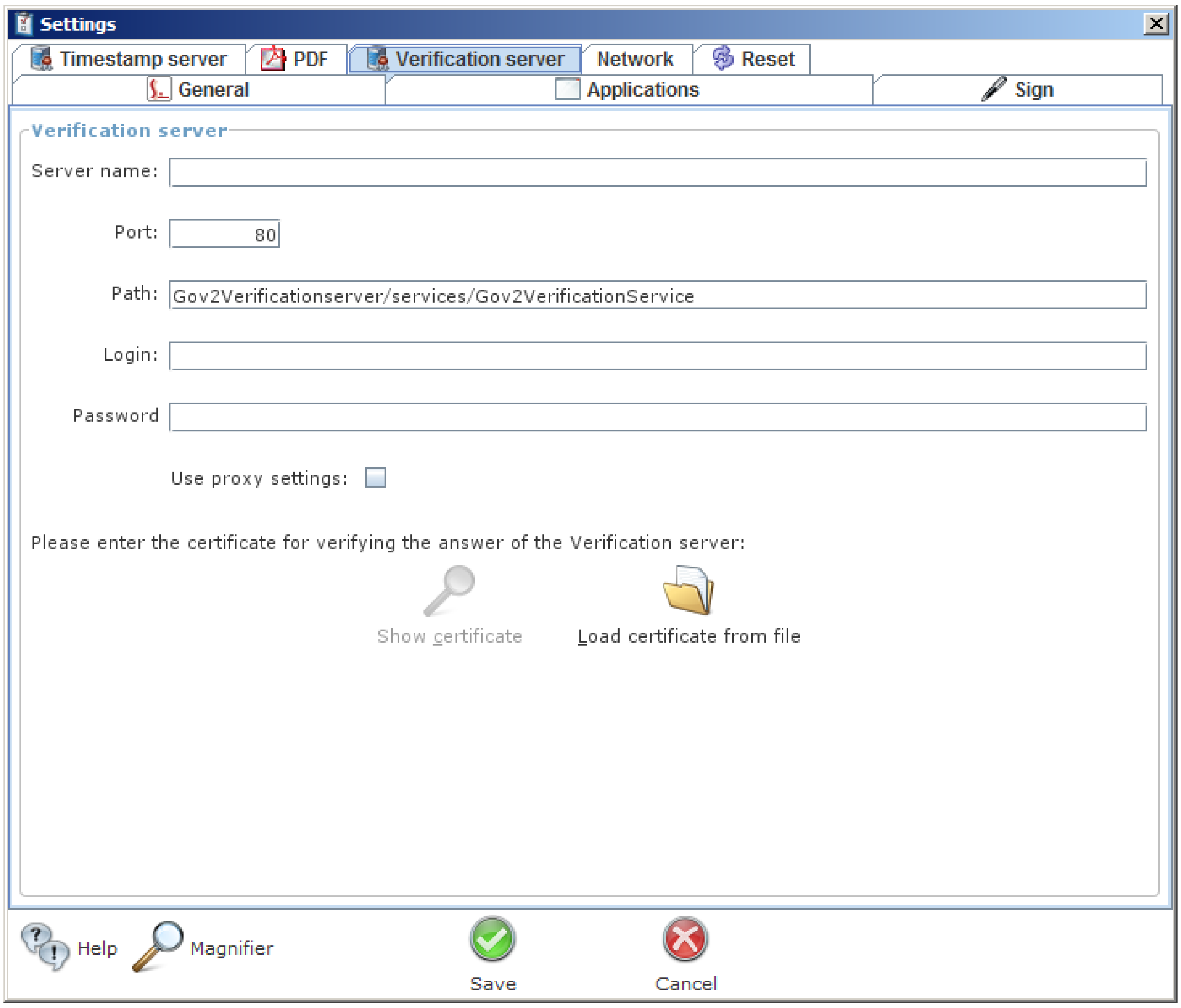
2.5.5 The Verification Server tab

Verification Server (XKMS responder)

You have to configure a Verification Server in order to use the Governikus Signer's "Verify" function. The Verification Server is needed in case the authenticity of an electronic signature is to be determined. You can check whether an electronically signed file is intact (integrity) and the signature really originates from the given person (authenticity). This process is described in the explanations chapter, section 2.7.9. You can get the required parameters for the following input fields from system administrator of XKMS responder.

- **Server name**: Fill in the name or IP address of the server, on which the Verification Server is reachable, for example www.example.com or 127.0.0.1.

- **Port**: Fill in the port number that is used for communication with the Verification Server. Usually this is port 80.
- **Path**: This path refers to the exact address on the Verification Server, for example `/Gov2Verificationserver/services/Gov2VerificationService`.
- **Login/Password**: If the Verification Server needs authentication data fill in your login name and password here.
- **Use proxy settings**: If the computer that runs your Governikus Signer is protected by a proxy server check this checkbox. The proxy server is configured on "The Network tab".
- **Load certificate from file**: This certificate secures the communication between your Governikus Signer and the Verification Server, so you can trust the Verification Server's responses. Click on the directory symbol and navigate to the directory that contains the file with the certificate. This certificate is provided by your Governikus administrator. The file must have the suffix `.cer` or `.crt`.
- **Display certificate**: Click this button to display the certificate in a separate window. This button is only active if a certificate was loaded. After loading a certificate the button is labelled with the certificate owner's name. You can either close the dialog with the OK button or save the certificate as a file with the "Save" button . Use the "Verify" button for an online verification of the certificate. The inspection sheet is displayed in a separate window.



Server name:
Port: 80
Path: Gov2Verificationserver/services/Gov2VerificationService
Login:
Password
Use proxy settings: ☐

Please enter the certificate for verifying the answer of the Verification server:

Figure 13: "Verification Server" tab

Shortcuts on this dialog

- Alt + c = the selected certificate is displayed
- Alt + l = load certificate from file

2.5.6 The Encrypt tab

Attention: This tab is not shown in this version of the Governikus Signer.

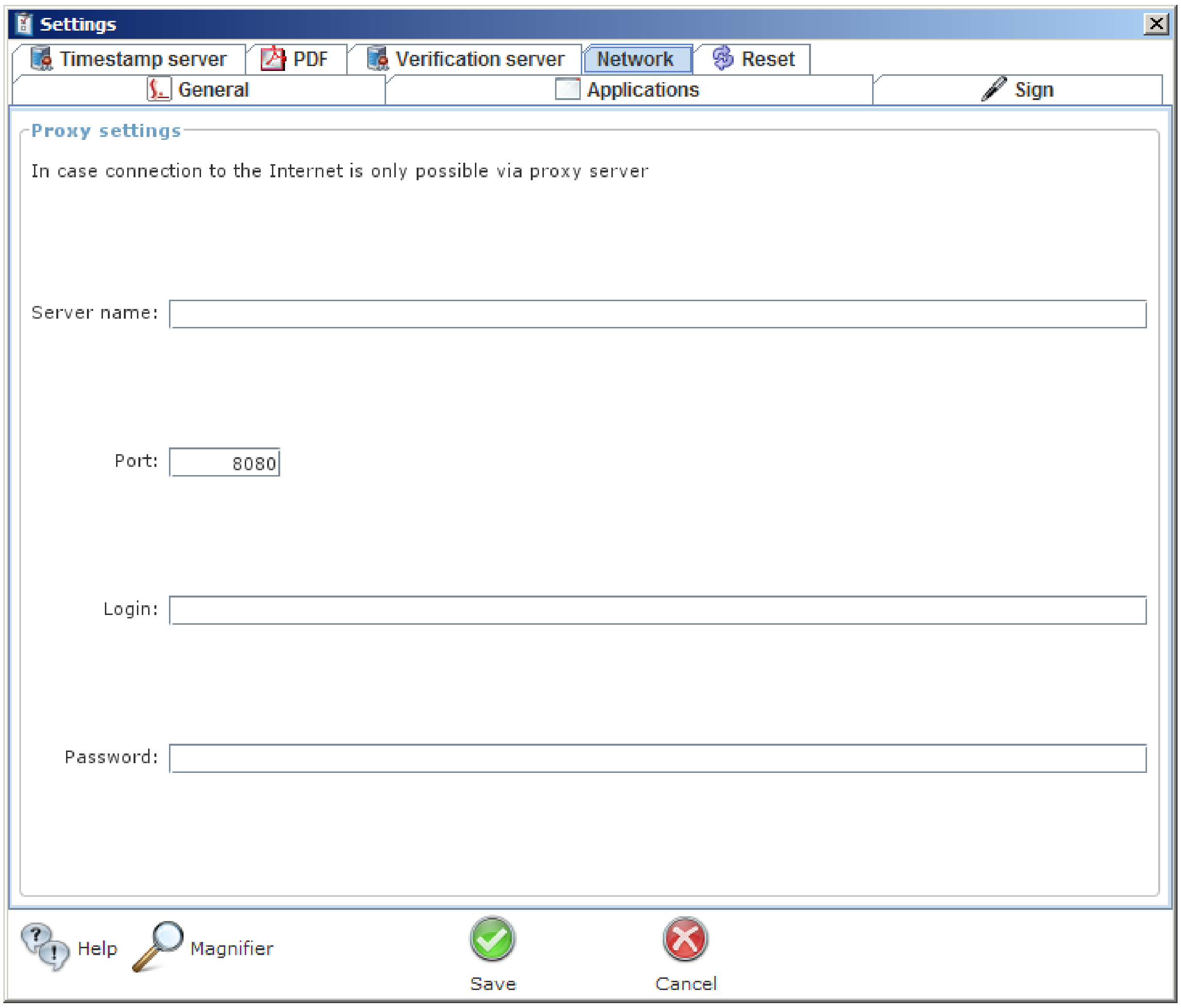
2.5.7 The Network tab

This tab offers configuring an update server and a proxy server.

Proxy settings

If a proxy server is operated between your computer and the internet you must enter the according access data here. All checkboxes of the same name on the various tabs of the settings dialog refer to this proxy server. If a proxy is already configured on your computer it settings are automatically retrieved and filled in. Otherwise, the configuration data for the proxy server is provided by your system administrator:

- **Server name**: Fill in the name or IP address of the proxy server, for example www.example.com or 127.0.0.1.
- **Port**: Fill in the port number that is used for communication with the proxy server. Usually this is port 80.
- **Login/Password**: If the proxy server needs authentication data fill in your login name and password here. Please mind that login and password are case sensitive.


Settings
Timestamp server | PDF | Verification server | Network | Reset
General | Applications | Sign

Proxy settings

In case connection to the Internet is only possible via proxy server

Server name:

Port: 8080

Login:

Password:

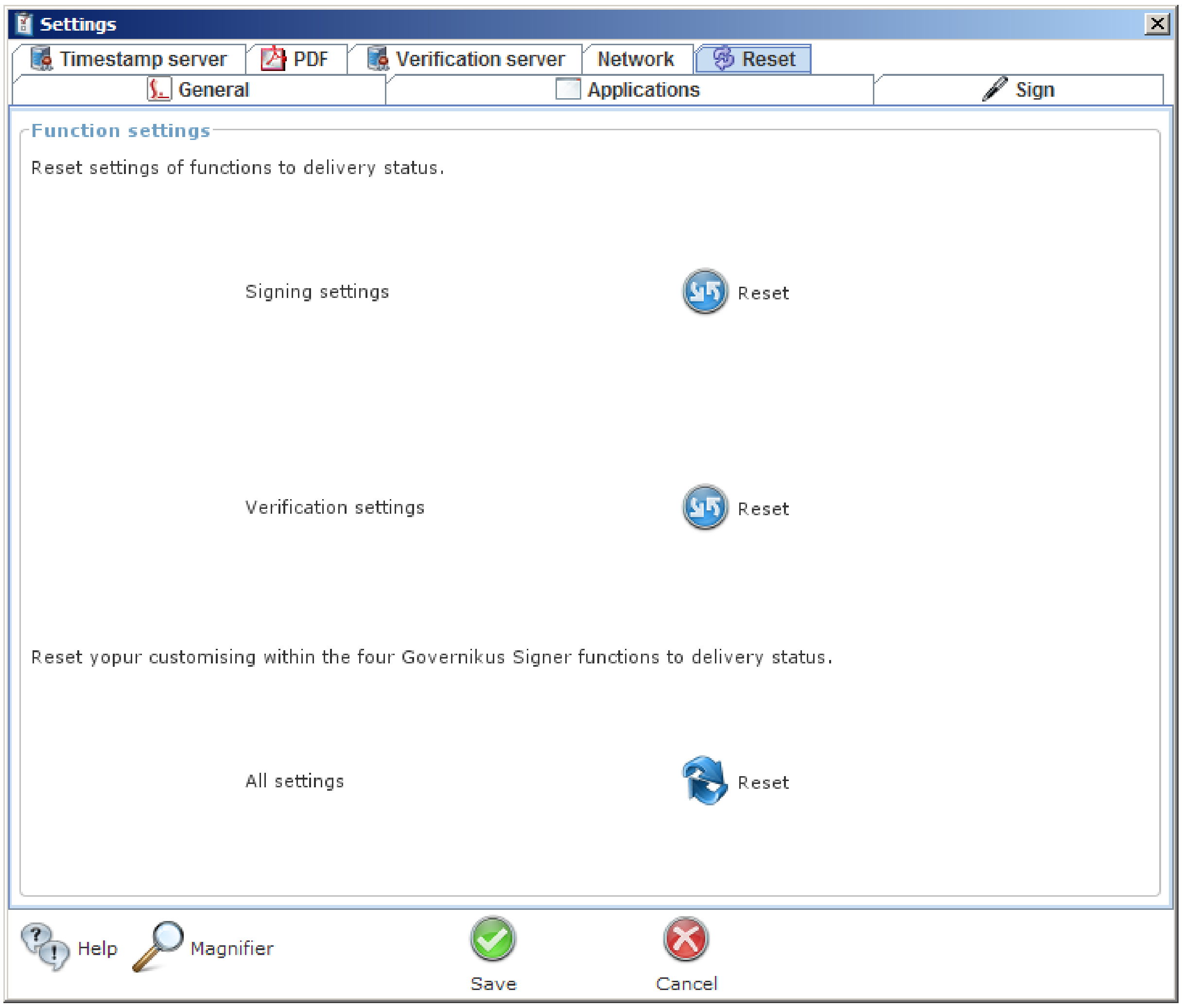
Figure 14: The "Network" tab

2.5.8 The Reset tab

The "Reset" tab allows resetting all or selected settings.

2.5.8.1 Function settings

- **Signing settings**: You can save your settings as default in the succession of dialogs in the "Sign" function. Thus these dialogs are omitted. You can reset your defaults here.
- **Verification settings**: This reset functions as is explained in the "Signing settings" (see above).
- **Reset all function settings**: This resets all of the above function settings. Afterwards the settings represent the state they had at delivery time of the Governikus Signer.



Reset settings of functions to delivery status.

Signing settings Reset

Verification settings Reset

Reset yopur customising within the four Governikus Signer functions to delivery status.

All settings Reset

Figure 15: The "Reset" tab

2.5.9 Export settings

All settings done on the tabs of the settings dialog are saved to the file `bos_signer_V2.xml` that is located here:

- For Windows: `C:\Documents and Settings\<login-name>`
- For Linux: `/home/<login-name>`

`<login-name>` stands for the name of your profile on the computer.

Administrators can export this file for example to a server that is available company-wide. You can then import this file via the main menus option "Extras" - "Import configuration file" that is explained in chapter "2.6.1". Thus it is possible to operate all Governikus Signer programs on all computers in the company with the same settings. Please mind the following notes to the single tabs of the settings dialog.

- The general tab: For the log file directory please mind that the drive must exist on the target computers.
- The Applications tab: Pleases mind that application configured in the processing tab must exist on the target computers in the same path.
- The Verification Server tab: All settings are taken over directly. The certificate is exported and imported as well.
- The Network tab: All settings are taken over directly.
- The Reset tab: This tab has no effect on the import.

2.6 The main functions

The Governikus Signer in this version has two main functions. Clicking on the appropriate symbol calls the respective function.

- **Sign**: Offers electronically signing one or more files.
- **Verify**: Enables the proof of integrity and authenticity of electronically signed files.

2.6.1 The Governikus Signer's menu bar

The menu bar offers the menus "File", "Extras", and "Help".

The File menu

The "File" menu offers the functions sign, verify and end. "End" closes the window and ends the Governikus Signer. The other four functions are subsequently explained in their own chapters.

The Extras menu

The "Extras" menu offers these functions:

- **Settings**: This option opens the "Settings" dialog that is explained in chapter 2.5.
- **Language**: You can use the Governikus Signer either in English or German. Changing the language immediately affects the user interface and can be switched back any time.
- **Import configuration file**: This option enables importing a file that already contains settings for the Governikus Signer. Thus it is, for example, possible to use the same settings company-wide. **Attention**: If you have already configured your Governikus Signer in the settings dialog

these changes are overwritten by this import. Offering a file with preconfigured settings is described in chapter 2.5.9.

- **Export configuration file**: This option enables saving the settings of your Governikus Signer to a file. The file has the suffix `.xml`. Information on which settings are contained in the exported file is described in chapter 2.5.9.
- **Reset settings**: Use this option to delete all your personal customising in the "Settings" dialog. The Governikus Signer's settings are reset to delivery status.
- **Reactivate disabled dialogs**: Certain dialog windows offer a checkbox labelled "Don't show this dialog again". If you select this option all dialogs are displayed again that were previously blinded out.

The Help menu

The help menu offers the following functions:

- **Help**: This option opens the online help.
- **About Governikus Signer**: This option opens a dialog window that contains information on version and vendor.

The following figure shows the entry dialog of the Governikus Signer.

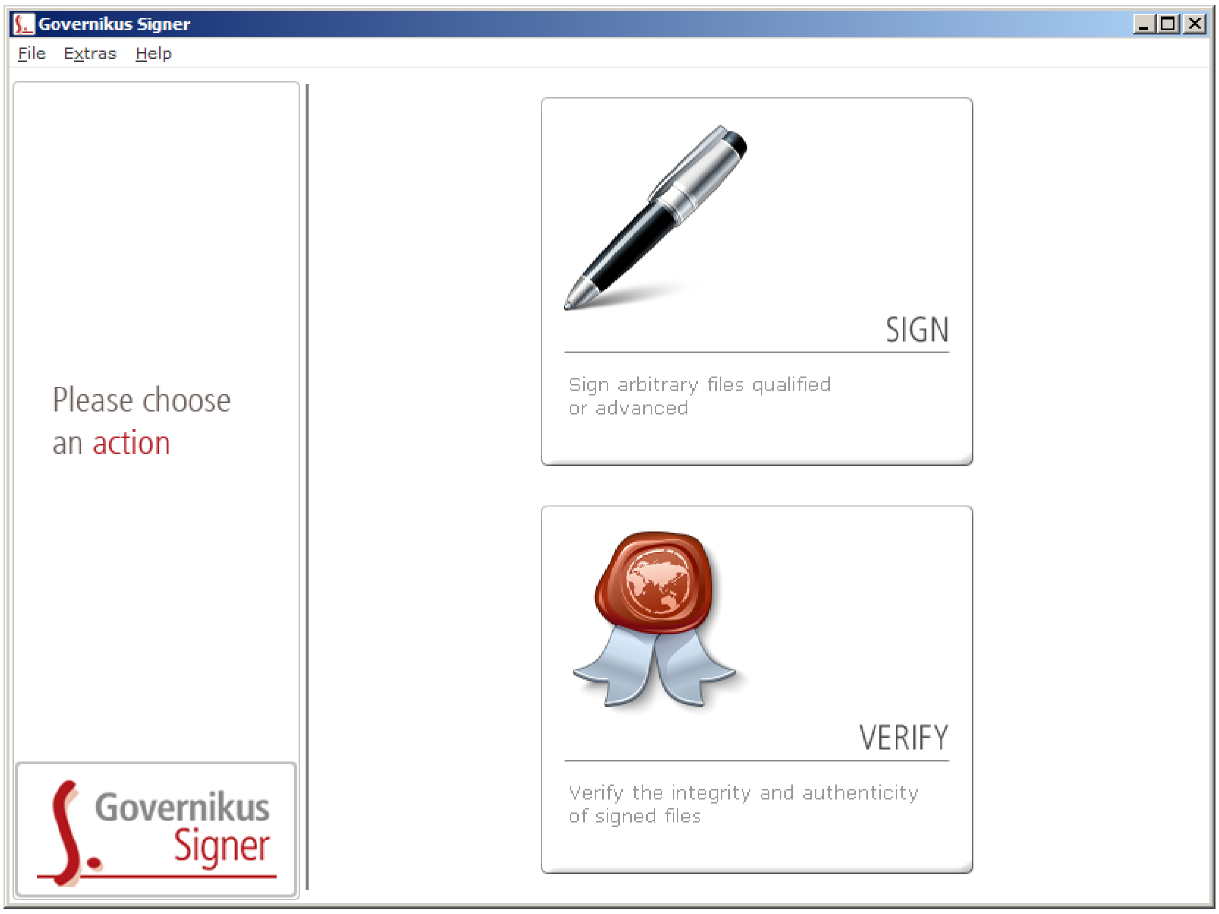

Governikus Signer
File
Extras
Help
Please choose an action
SIGN
Sign arbitrary files qualified or advanced
VERIFY
Verify the integrity and authenticity of signed files
Governikus Signer

Figure 16: Entry dialog of the Governikus Signer

2.6.2 Keyboard commands

This chapter lists the shortcuts that enable handling the Governikus Signer by keyboard.

Shortcuts on the entry page

- Alt + s = Sign
- Alt + v = Verify
- Alt + f = Opens the "File" menu
- Alt + x = Opens the "Extras" menu
- Alt + h = Opens the "Help" menu

General shortcuts

- Ctrl + F1 = Sign
- Ctrl + F2 = Verify
- Alt + F4 = Exit program
- Ctrl + 1 = Settings dialog
- F1 = Opens the online help

Shortcuts within selected functions

- Alt + (down arrow) = Select next navigation step
- Alt + (up arrow) = Select previous navigation step
- Alt + (right arrow) = Select next navigation step - dialogs, whose settings are marked as default, are omitted.
- Enter = corresponds to Alt + (right arrow)
- Alt + (left arrow) = Select previous navigation step - dialogs, whose settings are marked as default, are omitted.
- Alt + backspace = Back to entry page
- Alt + u = Reset button (if visible)
- F1 = context-sensitive help

Shortcuts within the certificate display

- Ctrl + s = Save certificate
- Enter = Close window
- Esc = Close window

2.6.3 Common features of dialogs

After selecting a function on the entry page the function dialog is displayed. It leads you from the function's settings to the function's execution. The structure of the pages is always the same.

- **Menu**: The menu bar in the uppermost line is always available. The menus are explained in chapter 2.6.1.
- **Left side**: The left side displays a button bar for selecting dialogs that you can call in succession. The last dialog offers executing the selected function with your settings.
- **Right side, above**: On the right side in the upper middle is a heading. This heading denotes the respective dialog step and corresponds to the label of the blue framed button on the left. This is for better orientation.

- **Right side, middle**: This area is for settings or starting actions.
- **Right side below**: This area displays buttons for navigation through dialogs, see next section "navigation".

Default settings

Some pages offer to save selected settings. Check the box "Save as default".

2.6.3.1 Navigating dialogs

- **Dialog selection**: Every dialog can be opened directly by clicking the respective dialog selection on the left hand side. The following figure shows the dialog selection as an example for the function "Sign". The currently selected page is framed blue. In case settings are still to be configured the step number is circled white, otherwise blue.

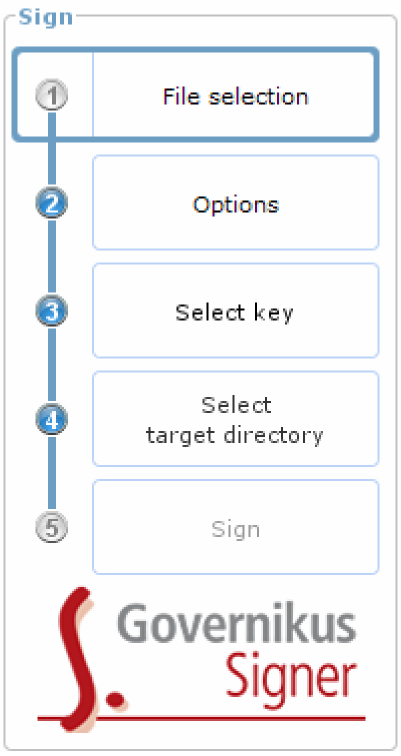


Figure 17: Dialog selection example "Sign"

- Use this button to call context-sensitive help
- **Magnifier**: This option creates a red frame in which the selected section is displayed enlarged. Click into the frame to leave the magnifier function.
- : Buttons with back and forward arrows are in the lower right side. Use these arrows to switch back and forth between dialogs. Dialogs are omitted that are marked as default (see above). In case required settings are missing on a dialog, the forward arrow is not usable.
- : The arrow that points left down is on the last dialog of a function and directly leads you back to the first page "File selection". Files that were not already processed on this last page

are kept in the file selection. Successfully processed files are not listed in the "File selection" dialog.

- ✖: The cancel button is displayed on every dialog and leads you back to the Governikus Signer's entry page.
- ✓: After processing of files is finished on a function's last dialog, the cancel button is changed to the OK button. This button as well leads you back to the Governikus Signer's entry page.

Note: All buttons, even when greyed out, have tooltips. Buttons are greyed out in case required settings are missing. If on a function's last page a button is greyed out, the tooltips gives the reason.

2.6.3.2 Selecting files

The file selection dialog is the same for all functions. The page's right side displays a list that is empty on first start. You can select any number of files from different directories. There are different ways to add files to the list.

1. Drag-and-drop

Mark one or more files in the file manager and, while the left mouse button is pressed down, drag your selection into the list of the Governikus Signer.

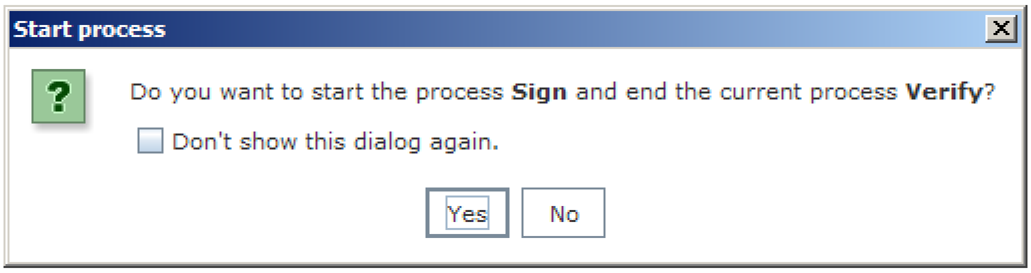
2. Button "Add file"

The button "Add file" opens a dialog window for file selection. Navigate to the directory, select the files and click on "Add". The file list now contains your selection.

3. Pass files per context menu

You can mark one or more files in the file manager and call the context menu by right clicking. In case the "File selection" dialog already contains files, you are asked in an extra dialog window whether you want to replace the existing selection or add the new files to the existing ones.

In case you have selected a certain process in the context menu, for example "Sign", and the Governikus Signer is currently displaying another process, for example "Verify", you are asked whether you want to exit this current process.


Start process

Do you want to start the process **Sign** and end the current process **Verify**?

☐ Don't show this dialog again.

Figure 18: Notification dialog for process change

Selecting several files at once

There are different possibilities to select several files at a time in the file manager or in the "Add file" dialog.

- **Select a list of files**: If you want to select a number of files that are listed one below the other, mark the uppermost with "Shift + left mouse click" and afterwards the lowermost file with "Shift + left mouse click" in that list. The selected files are now highlighted. You can now either drag these to the Governikus Signer's file list or use the "Add" button in the file selection dialog.
- **Select several files**: If you want to select several files that are not listed one below the other hold down the "Ctrl" key and select all required files by clicking on them with the left mouse button. The selected files are now highlighted. You can now either drag these to the Governikus Signer's file list or use the "Add" button in the file selection dialog.
- **Select all files**: If you want to select all files of a directory open the respective directory and mark all files with the shortcut "Ctrl + a". The selected files are now highlighted. You can now either drag these to the Governikus Signer's file list or use the "Add" button in the file selection dialog.

Remove selected files

You can remove one or more files from the file list of the dialog "File selection". You can select these files in the list by selecting them the way it was explained above in the section "Selecting several files at once". Afterwards click the button "Remove selected files".

List display

All selected files are displayed in a list. The columns have the following meaning:

- **File**: Displays the file name. The entire path and file name is displayed when pointing the mouse pointer to the file name. Double clicking displays the file.
- 👁: The eye symbol indicates that the file was already opened (see chapter 2.5.1, section "Number of files to be viewed"). The symbol is grey in case the file was not already viewed.
- 🔍: Click on the magnifier symbol to view the file. Since the resulting action is depending on the selected function, the appropriate behaviour is explained in the respective chapters of the functions. Note: Files selected for decryption cannot be viewed.

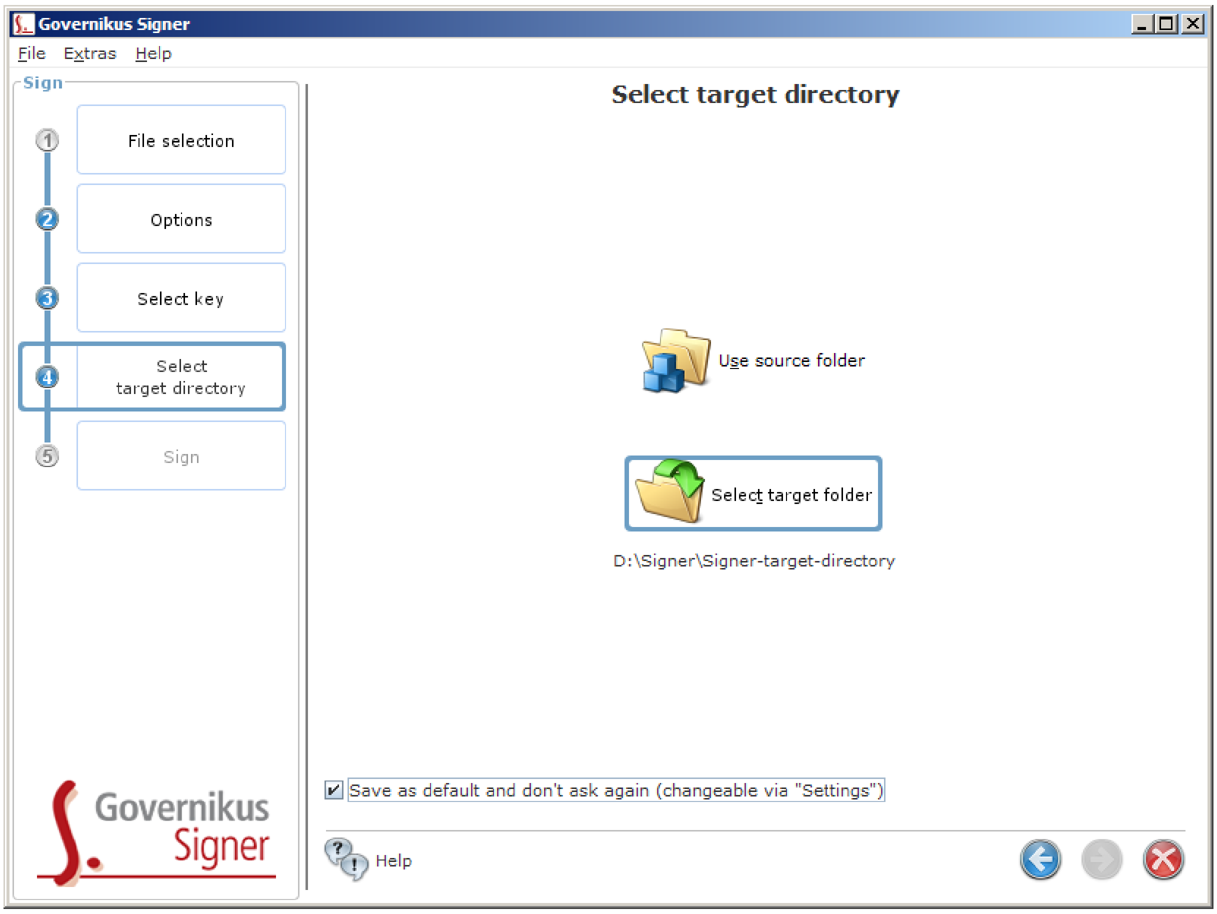
2.6.3.3 Selecting a target directory

The dialog "Select target directory" also exists for every function of the Governikus Signer. However, the function "Verify" displays this setting in the "Options" dialog otherwise it is to be found at the given name.

The target directory contains the files after execution of a selected function. The dialog offers two options. You can either use the source directory or select a new target directory. The selection is framed blue.

- **Use source folder**: This is the default selection. After executing the selected function the result files are saved to the same directory that contains the source files.
- **Select target folder**: This option opens the file chooser dialog. Here you can determine the target directory that will contain all result files. The path to the directory is displayed underneath the button "Select target folder".

In this dialog you have the possibility to save your settings. Check the box "Save as default and don’t ask again" on the lower left part of the dialog. The next figure shows the dialog with the selected checkbox.



Select target directory

Use source folder

Select target folder

D:\Signer\Signer-target-directory

☑ Save as default and don't ask again (changeable via "Settings")

Figure 19: Dialog "Select target directory"

2.6.4 Using the Sign function

Select this dialog to electronically sign files. A description of the electronic signature is given in chapter 2.7 section 2.7.4.

Please note: In this version of the Governikus Signer, as the **PEPPOL validation client**, the main function is the validation. To use the sign function with smartcard based qualified signatures, only German smartcards can be used.

Calling the sign function

On the entry page of the Governikus Signer click on "Sign" or use the shortcut "Alt + s" or "Ctrl + F1".

Calling the function with files

You can select one or more files in the file manager and select "Sign" from the context menu. In case the Governikus Signer is not yet started it will be started with your file selection.

2.6.4.1 Selecting files

"File selection" is the first dialog of the "Sign" function. Here you can select the files that you want to sign electronically. Different ways of selecting files is explained in chapter 2.6.3.2.

The file list

Line by line the file list shows the files that you have selected for signing. The columns have this meaning.

- **File**: Displays the file name. The entire path and file name is displayed when pointing the mouse pointer to the file name. Double clicking displays the file.
- : The eye symbol indicates that the file was already opened (see chapter 2.5.1, section "Number of files to be viewed"). The symbol is grey in case the file was not already viewed.
- : The magnifier symbol is in the last column of each row. Clicking on the symbol displays the file. If the file suffix is either `.txt`, `.tif`, or `.tiff` the Governikus Signer's own secure display opens that is explained in chapter 2.8. All other file suffixes lead to calling the program that is associated with the respective suffix, for example `.doc` files are displayed by the MS Word program. If you have selected "Trusted viewer" in the "Sign" tab of the settings dialog, this viewer is used for all files, as explained in chapter 2.8.

Shortcuts on this page

- Alt + a = Add file
- Del = Remove selected file
- Alt + t = Set focus to table

2.6.4.2 Setting options of the signing function

On the "Options" dialog you can select the signature format and further options.

Signature mode

The Governikus Signer supports several, internationally standardised formats for electronic signatures. These formats are explained in chapter 2.7 section 2.7.6.

- **Embedded according to PKCS7**: The file is signed according to the PKCS#7 standard. Exactly one new file is created. The file that is to be signed is embedded into a signature file. The newly created file has the same name as the original file; the file is extended by the suffix `.p7s`. Example: The file name `example.doc` becomes `example.doc.p7s`. The contained original file can only by viewed or extracted by an appropriate verification program, for example by the "Verify" function of the Governikus Signer.
- **Detached according to PKCS7**: The file is signed according to the PKCS#7 standard. Two files are created. One file is the original source file. The other file contains the electronic signature according to PKCS#7. For proving integrity and authenticity both files are needed. If for example the file `example.doc` is electronically signed with this option, the file named `example.p7s` is created, which contains the electronic signature. If the target directory is the source directory the `.p7s` file is saved there. In case you have set a separate target directory, the original source file and the `.p7s` file are saved there.
- **Sign PDF always inline**: With this format the signature is always saved within the PDF file. A file signed with this option has the suffix `_signed.pdf.` If this box is not checked, all PDF files are signed as any other file in the format that is selected above.

While selecting the signature format make sure that the selected format is permitted and accepted on the receiver's side.

Handling of existing signatures

This section only refers to PDF files that contain an inline signature, see section above. You can decide how to proceed in case the existing PDF file that is to be signed already contains one or more electronic signatures.

- **Replace all present signatures**: All existing signatures are deleted and replaced by your new electronic signature.
- **Sign serial**: In this case existing signatures are enveloped by your signature.

Default

As explained in chapter 2.6.3 you can save the settings on this page as default. If you use the navigation arrows in the lower right area of the dialog, this dialog is omitted in later signing calls.

2.6.4.3 Selecting a key

Use the "Select key" dialog to select the signature key that you want to use for electronically signing files.

- **Load key from file**: If you want to load a key from a file click on this symbol and navigate to the directory in your file system that contains the respective key. A keystore must be loaded whose filename ends with the suffix `.p12` or `.pfx`. A keystore contains a software certificate and the required pair of keys for the asymmetric encryption. For further information on asymmetric encryption read chapter 2.7.5. For software certificates please mind that authenticity of the signing person can only be verified in case the software certificate was issued by a trust centre and you have proved your identity by showing required identification documents. On loading the keystore file you are asked for the keystore's PIN.
- **Signature card**: This option is only displayed in case a card reader is connected and a signature card is inserted. Below the symbol the card readers name is displayed as recognised by the Governikus Signer. You can connect up to 10 card readers. In case you want to connect more card readers please read the document about system requirements. Usually you can create qualified electronic signatures or qualified accredited electronic signatures with a signature card.

Reread signature card

⚠	**Attention**: Please read the following section if a signature card is no longer readable by the Governikus Signer.

If a signature card is used by a signature application component of another vendor while the Governikus Signer is running, it may happen that the Governikus Signer can no longer read the signature card. This is, because the signature application component of the other vendor is blocking the card.

If you want to continue using the signature card with the Governikus Signer please proceed as follows:

- It is imperative to end the signature application component of the other vendor.
- Withdraw the signature card from the card read and insert it back again, or

- Click on the "Reset" button in the dialog section "Connected card readers".

The signature card is reread and afterwards you can again select keys from the signature card. The card reader however that holds the signature card is not affected by this action and operates as it did before.

After selecting the signature unit the available keys are listed by their corresponding certificates. A keystore or signature card may contain several keys. If so, you must select exactly one key from the list.

- : The displayed key belongs to a certificate that you can display by clicking on this magnifier symbol. You can either end the certificate display with the OK button or save it as a file with the save button. Use the certificate symbol button to execute an online verification of the certificate. The inspection sheet is displayed in a separate window.

Default

As explained in chapter 2.6.3 you can save the settings on this page as default. If you use the navigation arrows in the lower right area of the dialog in later signing calls, this dialog is omitted. However, the default is ignored in case the saved key is no longer available.

Attention: Only qualified electronic signatures - accredited or not - are the same as your legally binding, personal, handwritten signature.

As of now these electronic signatures can only be created with signature cards that are issued by an accredited certification authority according to the German Digital Signature Act.

Shortcuts on this page

- Alt + c = Display selected certificate

2.6.4.4 Selecting a target directory

How to select a target directory is explained in chapter 2.6.3.3. The target directory contains the electronically signed files.

Default

As explained in chapter 2.6.3 you can save the settings on this page as default. If you use the navigation arrows in the lower right area of the dialog in later signing calls, this dialog is omitted.

Shortcuts on this page

- Alt + s = Button "Use source folder"
- Alt + t = Button "Select target folder"

2.6.4.5 Executing the signing process

"Sign" is the last dialog of the signing function. The list contains all of your selected files. Click on the "Sign" button below the file list to start the signing function. The files are signed one by one. If you are signing with a signature card you must enter your PIN for each file.

The list display

The rows of the list-display consist of the following columns:

- **File**: Displays the file name. The entire path and file name is displayed when pointing the mouse pointer to the file name. Double clicking displays the file.
- : The eye symbol indicates that the file was already opened (see chapter 2.5.1, section "Number of files to be viewed"). The symbol is grey in case the file was not already viewed.
- : Clicking on the magnifier symbol displays the file. If the file suffix is either `.p7s`, `.txt`, `.tif`, or `.tiff` the Governikus Signer's own secure display opens that is explained in chapter 2.8. All other file suffixes lead to calling the program that is associated with the respective suffix, for example `.doc` files are displayed by the MS Word program.
- **Status**: This column displays the processing status of the file. The following status are possible:
  - **New**: The file wasn't yet processed;
  - **Working**: The file is currently processed;
  - **Finished**: The processing is finished;
  - **Error**: While processing the file an error occurred. If you point your mouse to that error, a tooltip shows the cause.
- **Result file**: After successful processing, this column holds the path and filename of the signed file.
- : The magnifier button displays the result file. The functionality of this button is explained in chapter 2.8.2.

Result file already exists

In case of detecting a file of the same name as the result file in the target directory, the "Target file exists" dialog is displayed. You can now choose whether to overwrite the existing file or rename the new result file. While renaming the file, the current date is added to the file name as prefix. However, you can as well cancel the execution.

Upon completion pass files to the following program

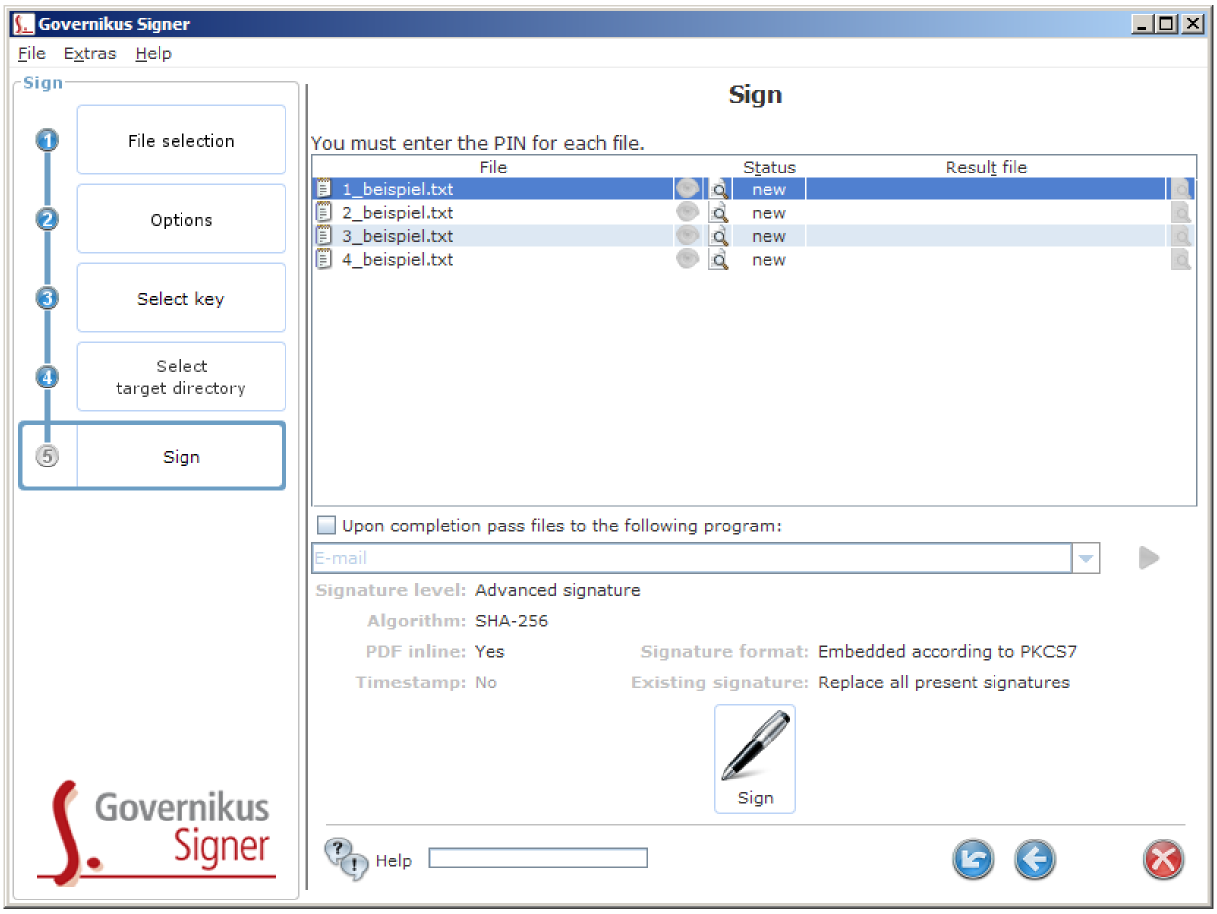
If you check this option you can select a program from the list below, which then further processes the result files. The selected program is directly invoked with the result files as parameters. In case the option is not checked the program list is greyed out. The program list contains all programs that you have added in the "Processing" tab of the settings dialog, see chapter 2.5.2.

Note: You can as well select this option after the files are processed. Check the box, select the program, and click the right-arrow symbol to start the further program execution.

Attention: **All** result files are passed to the succeeding program.

Last dialog of the "Sign" function

On the last dialog of the sign function below the file list and the succeeding program list the settings are summarised that you have selected for this signing pass. Click on the "Sign" button to start the process.



You must enter the PIN for each file.

File	Status	Result file
1_beispiel.txt	new	
2_beispiel.txt	new	
3_beispiel.txt	new	
4_beispiel.txt	new	

Upon completion pass files to the following program:

E-mail

Signature level: Advanced signature

Algorithm: SHA-256

PDF inline: Yes

Signature format: Embedded according to PKCS7

Timestamp: No

Existing signature: Replace all present signatures

Figure 20: Last dialog of the "Sign" function

Shortcuts on this page

- Alt + Enter = Button "Sign"

2.6.4.6 Special case batch signature card

Batch signature cards are special signature cards that enable signing several files while you must only enter your PIN once. However, you must enter your PIN again for every new selection of files.

2.6.5 Using the Verify function

What is verified?

Use this dialog to verify electronically signed files, certificate files and timestamp files. Verifying is explained in chapter "Explanations" section 2.7.9.

Calling the verify function

On the Governikus Signer's entry page click on Verify or alternatively use the shortcut "Alt + v" or "Ctrl + F2"

Calling the verify function with files

You can select one or more files in the file manager and select "Verify" from the context menu. In case the Governikus Signer is not yet started, it will be started with your file selection.

2.6.5.1 Selecting files

Select the files that you want to verify. Different ways of how to select files are explained in chapter 2.6.3.2.

The file list

Line by line the file list shows the files that you have selected for verifying. The columns have this meaning.

- **File**: Displays the file name. The entire path and file name is displayed when pointing the mouse pointer to the file name. Double clicking displays the file.
- 👁 : The eye symbol indicates that the file was already opened. The symbol is grey in case the file was not already viewed.
- 🔍 : The magnifier symbol is in the last column of each row. Clicking on the symbol displays the file. The display is explained in chapter 2.8.

Shortcuts on this page

- Alt + a = Add file
- Del = Remove selected files
- Alt + t = Set focus to table

2.6.5.2 Setting options of the verifying function

While verifying the signed file, the integrity is always checked locally. Thus it is detectable whether the file was altered after signing. You can configure the following settings on this page.

Verification methods

- **Online verification**: For checking the box "Do you want to execute an online verification" a Verification Server must be configured. The link on the right hand side leads you directly to the "Verification Server" tab in the settings dialog, see chapter 2.5.5. Online verification enables you to verify the authenticity of a signature card owner at a certificate authority for the respective electronically signed file. Evidence about validity of a certificate at signing time is detectable.
- **No online verification**: No online verification is executed at a certificate authority, if you don't check the option.

Target directory for inspection sheet

Selecting a target directory is done as explained in chapter 2.6.3.3.

Default

As explained in chapter 2.6.3 you can save the settings on this page as default. If you use the navigation arrows in the lower right area of the dialog, this dialog is omitted in later verifying calls.

Shortcuts on this page

- Alt + s = Button "Use source folder"
- Alt + z = Button "Select target folder"

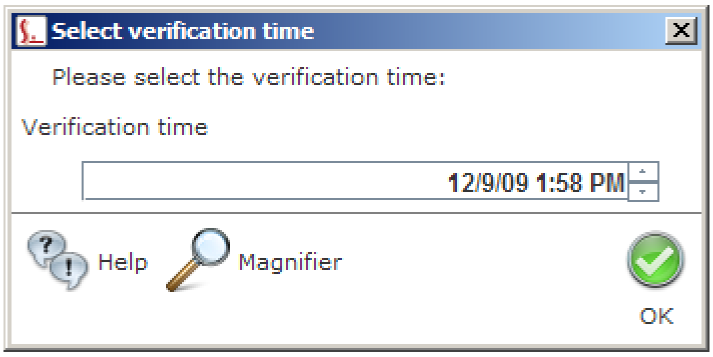
2.6.5.3 Executing the verifying process

"Verify" is the last dialog of the verifying function. The list contains all of your selected files. Click on the "Verify" button below the file list to start the verifying function. The files are verified one by one.

Note: Above the "Verify" button the Verification Server's name is displayed, as configured in the settings dialog. Prior to verifying please check whether this setting is still correct or if maybe it was manipulated.

Special case verifying certificate files

Certificate files are used for signing, encrypting, and decrypting. You can also verify these certificate files with the Governikus Signer's Verify function to get information about their validity. They usually have the file suffix `.cer` or `.cert`. If you load a certificate in the file selection dialog and click the "Verify" button on the last dialog page of the function, a dialog requesting the verification time is displayed. Please mind that this dialog is only displayed for the verification of certificate files.


Select verification time

Please select the verification time:

Verification time

12/9/09 1:58 PM

Figure 21: Dialog window "Select verification time"

The current date and time is preset as verification time. You can change the verification time. The verification process checks whether the certificate was valid at the given time.

- **OK**: Clicking OK processes the verification with the verification time you have filled in. If several certificate files are in the file list, the selected time is used for all these certificate files. The dialog is not shown again in this verification pass.
- **Cancel**: If you cancel the dialog with the X in the upper right corner of the dialog window the current time is used for verifying the certificate file. If you have uploaded several certificate files the dialog is displayed again for the next certificate file in the file list.

The list display

The rows of the list-display consist of the following columns:

- **File**: Displays the file name. The entire path and file name is displayed when pointing the mouse pointer to the file name. Double clicking displays the file.
- : The eye symbol indicates that the file was already opened (see chapter 2.5.1, section "Number of files to be viewed"). The symbol is grey in case the file was not already viewed.

- : Clicking on the magnifier symbol displays the file as is explained in chapter 2.8.
- **Status**: This column displays the processing status of the file. The following status are possible:
  - **New**: The file wasn't yet processed;
  - **Working**: The file is currently processed;
  - **Finished**: The processing is finished;
  - **Error**: While processing the file an error occurred. If you point your mouse to that error, a tooltip shows the cause.
- **Result file**: After successful processing, this column holds the path and filename of the inspection sheet. Visualising verification results:
  - ●: The green bullet with the OK hook shows that all inspections have a positive result.
  - ●: The yellow bullet with the exclamation mark shows that at least one inspection could not be executed.
  - ●: The red bullet with the cross shows that at least one inspection had a negative result.
- : The magnifier button displays the inspection sheet. Clicking on this symbol invokes the browser that is associated with HTML files on your computer.

Result file already exists

In case of detecting a file of the same name as the result file in the target directory, the "Target file exists" dialog is displayed. You can now choose whether to overwrite the existing file or rename the new result file. While renaming the file, the current date is added to the file name as prefix. However, you can as well cancel the execution.

Upon completion pass files to the following program

If you check this option you can select a program from the list below, which then further processes the result files. The selected program is directly invoked with the result files as parameters. In case the option is not checked the program list is greyed out. The program list contains all programs that you have added in the "Processing" tab of the settings dialog, see chapter 2.5.2.

	Note: You can as well select this option after the files are processed. Check the box, select the program, and click the right-arrow symbol ▶ to start the further program execution. **Attention**: **All** result files are passed to the succeeding program.

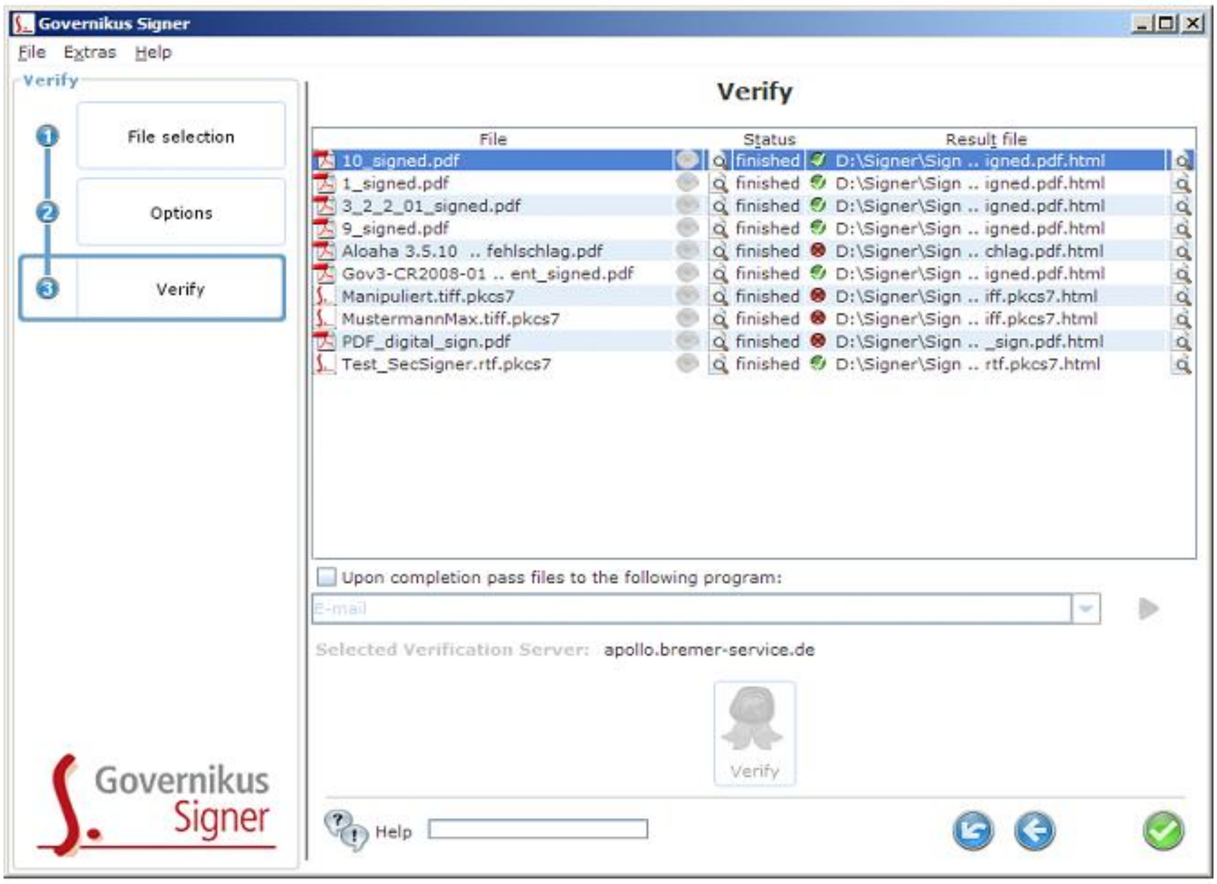
The inspection sheet

The verification's result is an inspection sheet, which is saved as HTML file in the target directory. The inspection sheet shows, which inspections had positive or negative results or whether an inspection could not be executed.

The inspection sheet is designed to be self explanatory and refferes to the results of the PEPPOL specification.

Last dialog of the "Verify" function

On the last dialog of the verify function, below the file list and the succeeding program list, path and name of the Verification Server is displayed. In case you haven't selected online verification the message "Online verification is deselected" is displayed. Click on the "Verify" button to start the process.


Governikus Signer

File Extras Help

Verify

1 File selection

2 Options

3 Verify

Verify

File	Status		Result file
10_signed.pdf	finished	✓	D:\Signer\Sign .. igned.pdf.html
1_signed.pdf	finished	✓	D:\Signer\Sign .. igned.pdf.html
3_2_2_01_signed.pdf	finished	✓	D:\Signer\Sign .. igned.pdf.html
9_signed.pdf	finished	✓	D:\Signer\Sign .. igned.pdf.html
Aloaha 3.5.10 .. fehlschlag.pdf	finished	✗	D:\Signer\Sign .. chlag.pdf.html
Gov3-CR2008-01 .. ent_signed.pdf	finished	✓	D:\Signer\Sign .. igned.pdf.html
Manipuliert.tiff.pkcs7	finished	✗	D:\Signer\Sign .. iff.pkcs7.html
MustermannMax.tiff.pkcs7	finished	✗	D:\Signer\Sign .. iff.pkcs7.html
PDF_digital_sign.pdf	finished	✗	D:\Signer\Sign .. _sign.pdf.html
Test_SecSigner.rtf.pkcs7	finished	✓	D:\Signer\Sign .. rtf.pkcs7.html

☐ Upon completion pass files to the following program:

E-mail

Selected Verification Server: apollo.bremer-service.de

Verify

Help

Governikus Signer

Figure 22: Last dialog of the verify function

Shortcuts on this page

- Alt + t = Set focus to table
- Alt + enter = Button "Verify"

2.7 Explanations

In the following the terms and backgrounds are explained that are important in the Governikus Signer's context. The explanations will deepen the understanding of the provided functions. Definitions and explanations in this chapter don't claim to be complete and cannot substitute legal advice.

Explanations in this chapter are sorted alphabetically since due to different usage scenarios of the Governikus Signer no other order made sense.

2.7.1 Authentication

Authentication on the one hand is proof of permission, for example by providing login and password and on the other hand it is also proof of identity, as for example with a passport. In case of an electronically signed file stating the authenticity of the signature card owner is possible, thus ensuring identity of the signing person.

2.7.2 Certificate authority

Issuing signature cards

A certificate authority (CA) issues signature cards. When requesting a signature card your identity must be proven, for example by the German Postident procedure. The signature card is then issued to the requestor and an unlocking process must be executed. Afterwards the signature card is valid for the given validity interval. On verification of an electronic signature the issuing certificate authority acknowledges the authenticity of the person, who has applied the signature.

2.7.3 Decryption

Decryption is used in case a file was previously encrypted. The chapter 2.7.5 explains encryption and decryption.

2.7.4 Electronic signature

An electronic signature always refers to exactly one file. It can be contained in the file itself or it can be created as an own file. An electronic signature for files is comparable to a seal that grants integrity and intactness of things or containers. The following four types of electronic signatures are differentiated, of whom only the latter two are comparable to your legally binding, personal, handwritten signature.

- Simple electronic signature (for example your handwritten signature, scanned and inserted as picture file)
- Advanced electronic signature (for example created with a software certificate)
- Qualified electronic signature (created with a signature card)
- Qualified electronic signature with accredited issuer (created with a signature card)

Please also read:

Law basics of qualified electronic signatures

http://www.gesetze-im-internet.de/sigg_2001/index.html

http://www.gesetze-im-internet.de/sigv_2001/index.html

Authenticity and integrity

Task of the electronic signature is to grant authenticity and integrity of files. After you have electronically signed a file, it is possible to determine whether this file was really signed by you (authenticity) and whether the content of the file was manipulated after signing or not (integrity).

How is an electronic signature created?

An electronic signature is created in three steps. In the first step a hash value is computed for the file, in the second step the hash value is encrypted and in the third step the certificate is added.

1. Computing the hash value

A function is used on the file that computes a unique value for the file. The function is called hash function and the value is called hash value. A hash value requires much less disk space than the file, for which it is created. Example for a hash value:

```
0D9C3ECDFBE036E1750DE82A7863F1E6B6AC336B
```

A hash value is unique for every file. If the same hash function is always applied to the same file than the resulting hash value is always the same. If the file is changed another hash value results from the computation. Hence, a hash value is unique for a file and determines the integrity of a file. As long as the hash function's result is always the same hash value the file was not manipulated.

2. Encrypting the hash value

For encrypting the hash value a so-called asymmetric pair of key is used (for encryption see chapter 2.7.5). It consists of a private (secret) and a public key. The private key is only contained on the signature card and cannot be removed from there. The public key can be accessed by everyone. The private key is used to encrypt the hash value. To do so, a program like the Governikus Signer computes the hash value and passes it to the signature card. Within the signature card the hash value is encrypted and then the encrypted hash value is passed back to the program. In order to prevent misuse of the signature card the personal identification number (PIN) is requested prior to encryption. Only on correct PIN entry the encryption is started.

3. Adding the certificate

After passing back the encrypted hash value to the program the certificate is copied from the signature card and added to the encrypted hash value. It contains the signature card owner's name, the public key and the certificate authority (CA, see chapter 2.7.2) that issued the signature card. Furthermore the time is added, at which the encryption was executed.

Signed file

The parts explained above - encrypted hash value, time of encryption and certificate with public key - constitute the electronic signature. The electronic signature can be contained in the signed file itself, as is done in PDF files. On the other hand the signature can contain the signed file. This signature is then called enveloped. In case the signature is contained in an extra file it is called detached. The certificate can be traced to the certificate authority. The certificate authority acknowledges on request the signature card owner's identity which also proves the authenticity.

Attention: The content of a file that was "only" electronically signed and is not encrypted can be viewed by a third party. The electronic signature proves authenticity and integrity but without encryption the content is not secret.

2.7.5 Encryption

On encrypting a file it is transformed from its original state to a secret representation. This can be done with all sorts of files like text-, picture- or program files. Text- or picture files are no longer viewable after encryption; program files are no longer executable after encryption. Encryption ensures that a third party has no access to content or functionality of a file. For reversing this transformation the file must be decrypted. The encryption process for files is either asymmetric or symmetric.

Asymmetric encryption

For asymmetric encryption a pair of keys is required. This pair consists of a private and a public key. The private key is never handed out; the public key can be passed to all of your business partners. Business partners exchange their public keys. If a file is to be encrypted the public key of the business partner is used. Then, only this particular business partner is able to decrypt the encrypted file with his secret, private key.

- Advantage: since only the receiver can decrypt the files with his private key, the public key can be sent without danger to other recipients.
- Disadvantage: Asymmetric encryption is by far more time consuming, since the process (the algorithm) is very complex.

Symmetric encryption

The symmetric encryption uses only one key for encryption and decryption.

- Advantage: This procedure is by far faster than the asymmetric process.
- Disadvantage: If the symmetric key is sent and is intercepted by a third party everybody that has this key can decrypt messages that were encrypted with this key. Hence, it is also possible to decrypt a file, change it and encrypt it again.

The S/MIME standard that is used by the Governikus Signer for encryption and decryption contains both methods. The file is first encrypted with the fast symmetric encryption. The required symmetric key is created by the Governikus Signer. For each file that will be encrypted a new key is created. The symmetric key is then encrypted with the public key of the receiver and added to the encrypted file. The receiver now uses his private key to decrypt the symmetric key and then uses the symmetric key to decrypt the file. In case an encrypted file is to be sent to various recipients the symmetric key is encrypted step by step with all public keys of the recipients. All resulting encrypted symmetric keys are then added to the encrypted file and thus all recipients are able to decrypt the sent file. For the user of the Governikus Signer this procedure is completely automated and only pressing the "Encrypt" button is required.

2.7.6 Formats of electronic signatures

The Governikus Signer supports various formats of electronic signatures. These formats are internationally standardised signature and file formats. The supported formats are:

PKCS#7

PKCS#7 is by far the most spread format. The acronym PKCS stands for public key cryptography standard and is part of the IETF standard Cryptographic Message Syntax Standard (CMS). Currently 15 PKC standards are described (PKCS#1 to PKCS#15) and PKCS#7 defines a standard for electronic signatures. This standard has a very precise specification and every software program for applying electronic signatures must strictly comply with it. Thus it is secured that programs of different vendors that are able to verify files that are electronically signed according to PKCS#7 will come to the same verification results.

PDF with embedded PKCS#7 signatures

This format is a variant of the above described PKCS#7 format. Here the PDF file remains displayable by every PDF reader program because the signature is listed in an own tab of the reader.

2.7.7 Signature cards

A signature card usually has the same format as a credit card and contains a chip. The chip usually contains three certificates, but more can be possible.

Certificates

Every certificate contains, among others, information on the owner (name, first name), the validity interval (start date and time until end date and time), the issuer (for example TeleSec of T-Systems), a fingerprint (for fast identification of a certificate's public key) and the key usage.

The three different certificates have among others an own serial number, an own fingerprint and different key usages. The three most common key usages are:

- keyEncipherment, dataEncipherment: A certificate with this key usage is used to encrypt or decrypt files.
- nonRepudiation: This certificate is used for signing files. Here the electronic signature is created.
- digitalSignature: This certificate is used in case the owner of a signature card wants to authenticate himself.

2.7.8 Timestamp

An electronic signature usually contains the point of time of the signing. The signing software can use the local system time. Since this time can be arbitrarily set by the user it cannot be trusted. An external timestamp of a trusted provider or, in case a legally binding time is requested, a qualified electronic timestamp of an accredited timestamp provider will help. Technically this external timestamp is a further signature file that signs the signature file for which it was requested. This second signature (timestamp file) contains a trusted point in time. The timestamp says: The timestamp provider acknowledges that the signature file existed at the given time, noted in the signed timestamp file. For example this service is provided by the AuthentiDate International AG. The timestamp of a certified timestamp provider has an electronic signature that is as well verifiable.

2.7.9 Verifying

Verifying is a process wherein an electronically signed file is checked for authenticity and integrity. The public key, which is usually contained in the certificate of the electronic signature, is used to decrypt the hash value. After re-computing the hash value it can be compared to the decrypted hash value. If they are the same the integrity of the signed file is proven. For verifying the authenticity, that is the identity of the person who claims to have signed the file, the certificate is sent to the online verification of the certificate authority. For the Governikus Signer this is done via a Verification Server, which is configured in the respective tag of the settings dialog. The certificate is passed to the Verification Server and further on to the issuing certificate authority by secured communication.

Certificate verification

The certificate authority verifies the certificate with respect to originality and validity. Validity in this context does not denote the validity interval of the certificate since it is contained in the certificate itself. The validity of certificate can be revoked before its validity interval expires, in case the owner has reported the certificate stolen, or if he fears the card and PIN are accessed by third party.

2.8 Secure display

The secure display is called for files with the suffix `.p7s`, `.txt`, `.tif` and `.tiff`. Tiff files can be manipulated. Please read this chapter prior to signing tiff files so that you can than examine potentially insecure files. Txt files as well can contain characters that can not be securely judged. Please read the next chapter about txt files. Electronically signed files with the suffix .p7s can be displayed by the secure display as well. Please read the chapter "Display file". How to proceed with tiff files is explained in chapter 2.8.3.

2.8.1 Secure txt display

The secure display can show txt files in UTF8 format unambiguously and without active or hidden content. Since the UTF8 coding allows non-displayable characters a subset of these UTF8 characters is implemented. The display shows each of these characters of the subset. In case of non-displayable UTF8 characters, the text file is not shown and a warning dialog is displayed instead.

2.8.2 Display files

In the functions described n the subchapters of chapter 2.6 the button with the magnifier can display files that are still to be processed (exception: decryption) as well as result files (exception: encryption). The display has this structure:

- **File menu**: In the upper left corner is the file menu. You can close this display and return to the Governikus Signer.

In a frame below the following data about the file is given: name, path, size, and signature format. Below, the display dialog is divided as follows:

Left side of the dialog window - upper part

With a simple signature these entries are shown one below the other:

- **First entry - signed content**: This entry depends on the file type. For all file types you can call a context menu here for saving or displaying the file (see next section "Left side of the dialog window - lower part ")
  - If the file is of type .txt "Text document" is displayed, the right side shows the text.
  - If the file is of type .tif or .tiff "TIFF document" is displayed, the right side displays the picture.
  - If the file is of type .pdf "PDF document" is displayed, the right side shows information about the file.
  - For all other file types the filename and suffix is displayed. The right side displays a text that informs you that no secure display is available for this file type.
- **Second entry - signature information**: If you select this entry the right side displays the name as found in the certificate, time of signature creation and the algorithm used for the hash value. The row "integrity" shows the result of the mathematical signature verification. A green circle with a white hook indicates that the document was not changed since signature creation.
- **Third entry - signature certificate**: This entry contains the signature certificate's owner, called subject name. The right side displays the most important details of the signature certificate. You can save the certificate via the context menu.

	Note: A validation whether the certificate of the signee is valid and not revoked is not processed.

In case the document contains several signatures the correlation of the signatures are displayed as a tree structure showing the context of signatures to signed documents.

Left side of the dialog window - lower part

This part contains these buttons:

- **Display**: This button calls the program that is associated on your computer with the document type and displays the file.
- **Save as**: This button opens a dialog window for saving the file. The file is automatically extended by the file suffix `.tmp`. If you change the suffix to it original name you can open the file with the program associated with this file type on your computer.

2.8.3 What is TIFF?

TIFF stands for **T**agged **I**mage **F**ile **F**ormat and describes picture data. Since there are various levels of handling picture data in tiff there is not only one representation. New or further developed tiff formats can be registered and published at ADOBE Inc. Today tiff is a widespread picture format, whose most common versions can be displayed by almost all picture viewers.

Note: The tiff specification (Adobe TIFF™ Revision 6.0) is receivable at:

http://partners.adobe.com/public/developer/tiff/index.html.

Structure and content of tiff files are explained here.

The complexity of the tiff specification enables displaying nearly lossless pictures and content. This complexity however offers possibilities to integrate hidden or active content.

The trusted tiff viewer of the Governikus Signer offers functionality to display content unambiguously. The user can thus be sure to view a file without hidden or active content. If hidden or active content is contained, it is noted in respective warning dialogs.

The secure tiff viewer has its own navigation. Here all picture and additional information can be displayed. Further functions enable colour changes to exclude potential manipulations.

Tiff files are recognised by their suffix. The suffixes `.tif` and `.tiff` are allowed. In the following all supported tiff formats are listed. Supported compression formats are only listed as far as the respective colour format is supported.

Supported colour formats (PhotometricInterpretation)

- WhiteIsZero
- BlackIsZero
- RGB

RGB paletteSupported compression formats (Compression)

- CCITT Group 3 1-dimensional modified Huffmann run length encoding (CCITT 1D)
- CCITT.4 bi-level encoding (Group 3 FAX 1D/2D)
- CCITT.6 bi-level encoding (Group 4 FAX)
- LZW
- PackBits
- Uncompressed

Supported colour depth (BitsPerSample)

- **Black and white pictures:** Here, only one bit per picture pixel is supported

- **Grey scaled pictures:** Only 4 or 8 bit per pixels are supported
- **Colour pictures**: Only RGB colour coded tiff pictures are supported. RGB colours must be set directly via a colour map. Colour information (red, green, blue) of a pixel may only have 8 bits, since with higher resolution the correct display of the pictures is not possible. With 16 bit on the colour palette it is checked, whether contrasts exist that are not displayable in true colour.

2.8.4 Structure of the secure tiff viewer

On opening a tiff document it is displayed in the trusted tiff viewer within the scope of the Governikus Signer, if at all displayable.

Note: Please mind that the display is parted into two sections when viewing PKCS#7 files. The upper part contains the usual information about the document whereas the lower part is reserved to the trusted tiff viewer.

The trusted tiff viewer is divided into the following areas

Navigation (including preview)

This area (left in the display) contains a table of contents for the picture. Besides the document name, all existing pictures of the tiff file are listed. When opening the trusted tiff viewer the first found picture is displayed by default. By changing between the content listing, different pictures as well as picture and additional information can be shown. The picture and additional information is viewable by clicking the file name. The lower part of the navigation area contains a preview area of the currently selected file.

Symbol- / Toolbar

On the lower end is the symbol- and toolbar of the trusted tiff viewer. The symbols open further functions. The symbol- and toolbar is only active if a picture is selected in the navigation area. If picture and additional information is selected the symbol- and toolbar is greyed out.

The symbols execute further functions within the trusted viewer. Their functionality is explained in the following chapters.

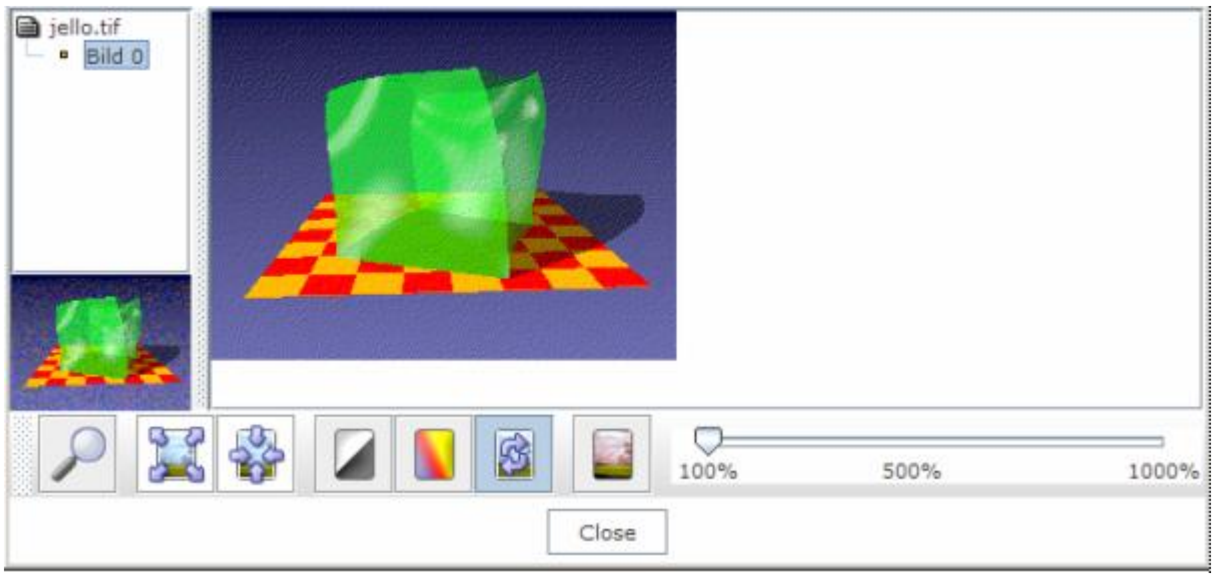


Figure 23: Structure of the trusted tiff viewer

2.8.5 Functions and menus of the trusted tiff viewer

The picture itself as well as invisible, descriptive contents of tiff documents belongs to the trusted tiff viewer. Opening a tiff documents displays the trusted viewer for the first found picture, which is then displayed.

Different functions can be applied to the selected picture via the buttons of the symbol- and toolbar to change the display of the picture. This function will help to display content that may be contained but is not visible in the normal display.

2.8.5.1 Changing the picture's display

The following functions are listed in the toolbar.

Magnifier

The magnifier enlarges parts of the picture. Clicking the magnifier button displays a frame in the picture with enlarged content. The frame is moveable with the mouse.

Note: Please mind that each pixel is only displayable in case the scaling is higher than 300%. Use the zoom function. If the scaling is below 300% pictures are anti-aliased that means that it is not possible to display each pixel separately.

Display size

Use these tools to adjust pictures to the window size. The picture is enlarged or reduced to fit the window.

Normal display

This tool changes the picture's size back to its original.

Threshold sliding

Activating this function displays an additional bar that is scaled from -255 to +255 and has a start button. Change the threshold by moving the sliders or via the start button.

A threshold is a value to define limits in raster pictures e. g. when converting black and white limits to binary values.

The threshold always has the same amount of colour channels as has the original picture. Coloured pictures are displayed in colour since this display offers more information than a black and white picture. The result is that contrasts that could usually not be distinguished with the naked eye are made visible by threshold sliding.

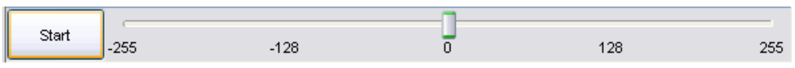


Figure 24: Scale for threshold sliding

Sliding colours

An additional sliding area is displayed that consists of three sliders for red, green, blue. The functionality is as explained above for the threshold sliding. Here even red-green handicapped persons may see smallest differences when moving the sliders or using the start button. This function is not available for black and white pictures.

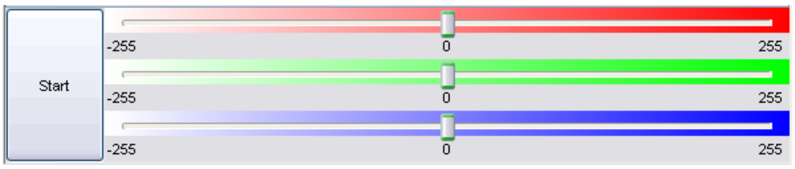


Figure 25: Scales for sliding colours

Normal view

This button leads you back to the original view of the picture in the 1:1 scaled display as well as switching off the magnifier and returning to the original colours.

Wrong colours display

Tiff documents may have their own colour-palettes and -tables. Using this function leads to randomly assigned new colours for the picture, which may make information visible that was previously hidden because of low contrasts. This function is the faster way than searching for appropriate thresholds. Creating wrong colours is done by selecting the required number of colours from a fixed order of colour values. This order is defined and determines the succeeding colour for all colours in a set of 15.357.184.

An advanced algorithm ensures that the colours are looking most different to differentiate colours and make the display visible for the naked eye.

Zoom function

The zoom reaches up to 1000%. Use this function to exclude anti-aliasing and to make single pixels visible. The zoom is only available in normal view.

The following picture shows the toolbar.

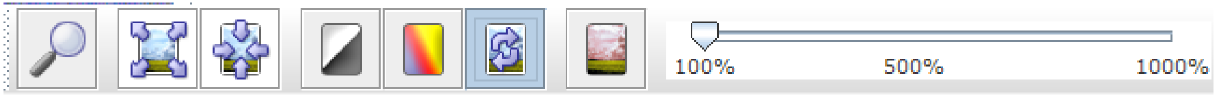


Figure 26: Symbol- and toolbar of the trusted tiff viewer

	Note: The slider can be operated with the mouse or automatically with the start button. It may however occur that this happens too fast or in too big steps. You can click on the respective slider area with the intended slider to activate it (activation is indicated by a dashed frame). Now use the cursor key's left- and right-arrow to initiate a single step sliding.

2.8.5.2 Picture information, unreferenced data areas and picture borders

Besides the picture itself, tiff-pictures can contain other information that is not visible on first sight. These may serve further description but they can as well contain active content. In order to exclude manipulations this additional information is also displayed. Click on the document's name in the navigation area for an overview of picture- and additional information. Two tabs show this information:

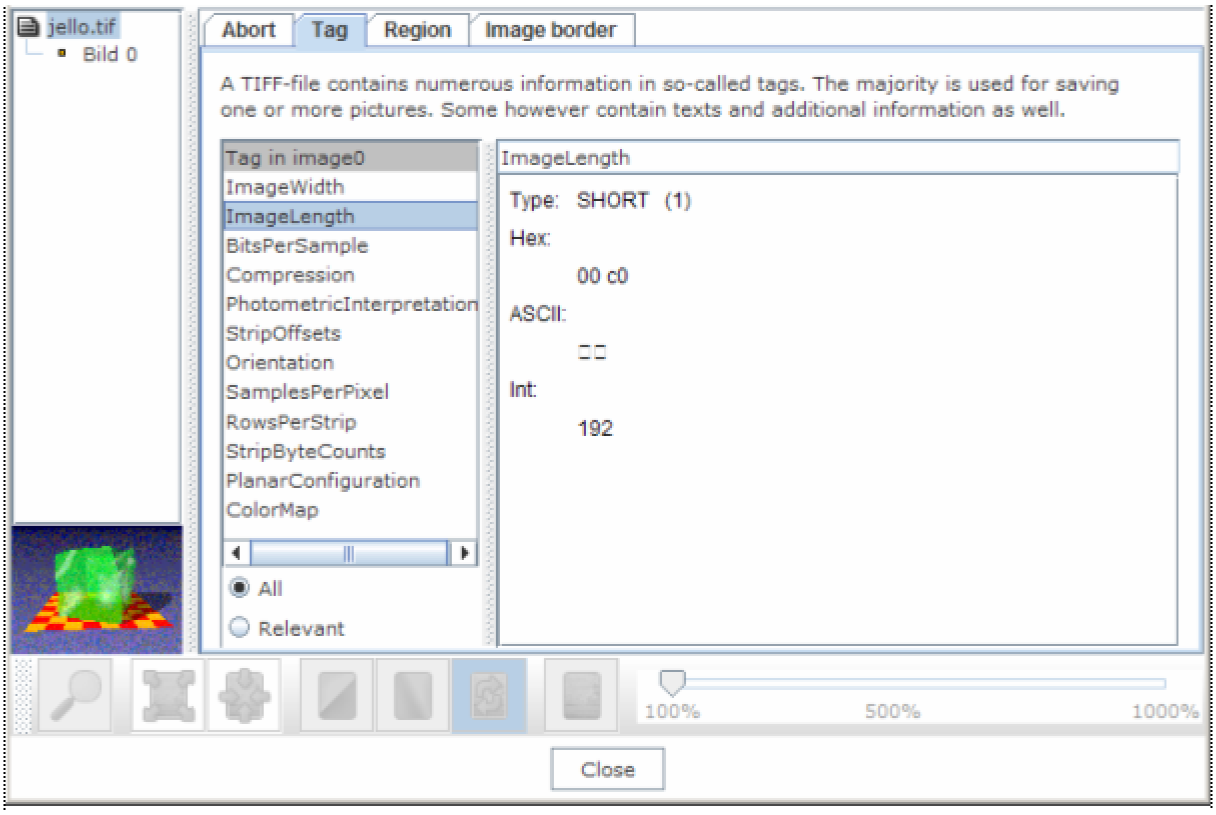
- "Tag" tab
- "Region" tab

Tag tab

The tag tab displays all tags of the selected tiff document. These tags can contain meta- or additional information that is saved within the picture. Usually they are for saving pictures, but they may as well contain further information. The overview to the left lists all tags found by default in a list. If a tag is selected details are listed on the right. The following information is displayed:

- **Type**: Type information on the tag
- **Hex**: Type in hexadecimal representation
- **ASCII**: Type representation in plain-text
- **Int**: Type representation as integer (if possible)

Unknown tag and tags of type ASCII are ranked as suspicious since further content maybe hidden. Use the radio buttons below to decide whether all or only suspicious tags will be displayed.



A TIFF-file contains numerous information in so-called tags. The majority is used for saving one or more pictures. Some however contain texts and additional information as well.

Tag in image0
ImageWidth
ImageLength
BitsPerSample
Compression
PhotometricInterpretation
StripOffsets
Orientation
SamplesPerPixel
RowsPerStrip
StripByteCounts
PlanarConfiguration
ColorMap

(● All) ○ Relevant

ImageLength

Type: SHORT (1)
Hex:
00 c0
ASCII:
□□
Int:
192

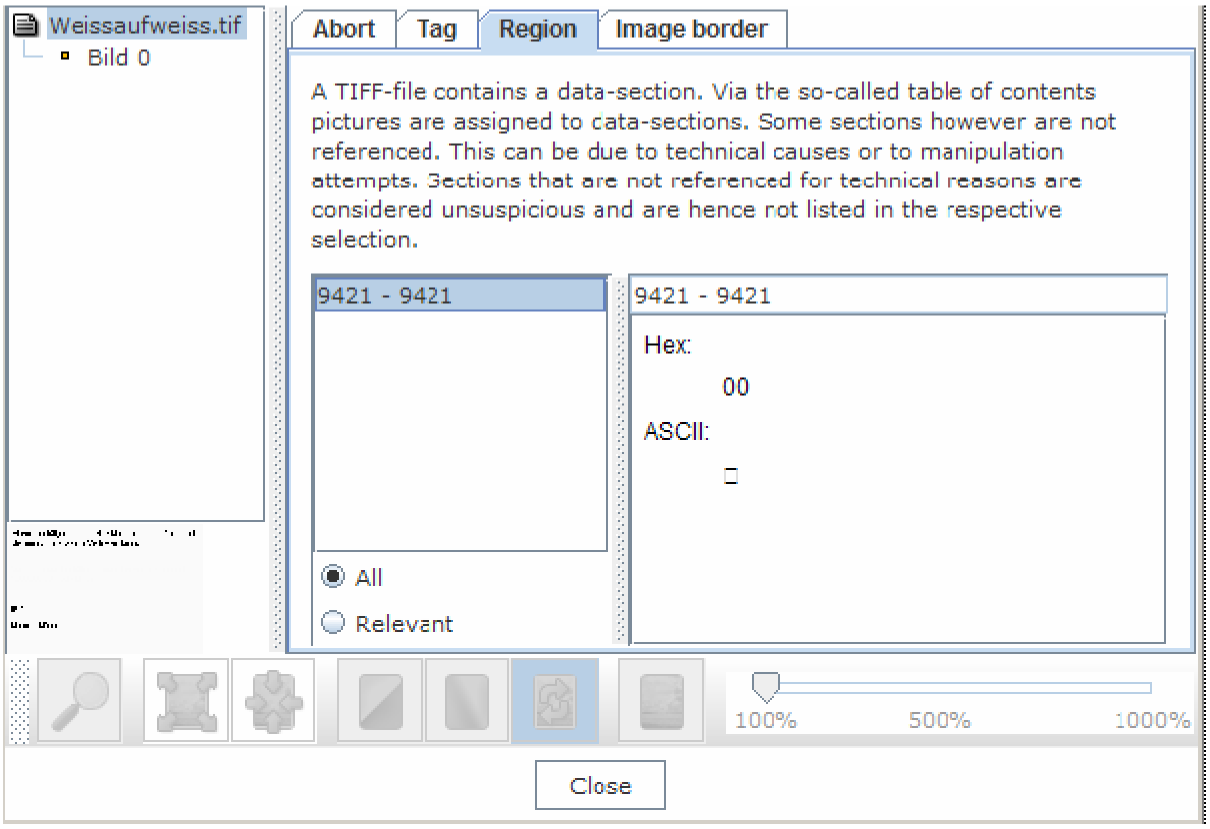
Figure 27: Secure display of picture information

Unreferenced regions

Besides tags so-called regions are referenced in data areas. These data areas are like tables of contents. Every region of the tiff document is referenced here. In the "Region" tab unreferenced regions are listed. After selecting a region details are shown on the right. The following details are displayed:

- **Hex**: Region in hexadecimal representation
- **ASCII**: Representation in plain-text

Use the radio buttons below to decide whether all or only relevant regions will be displayed. Unreferenced regions that are technically caused are taken as unsuspicious and not displayed in the "relevant" selection.



A TIFF-file contains a data-section. Via the so-called table of contents pictures are assigned to cata-sections. Some sections however are not referenced. This can be due to technical causes or to manipulation attempts. Sections that are not referenced for technical reasons are considered unsuspicious and are hence not listed in the respective selection.

9421 - 9421

9421 - 9421

Hex:

00

ASCII:

□

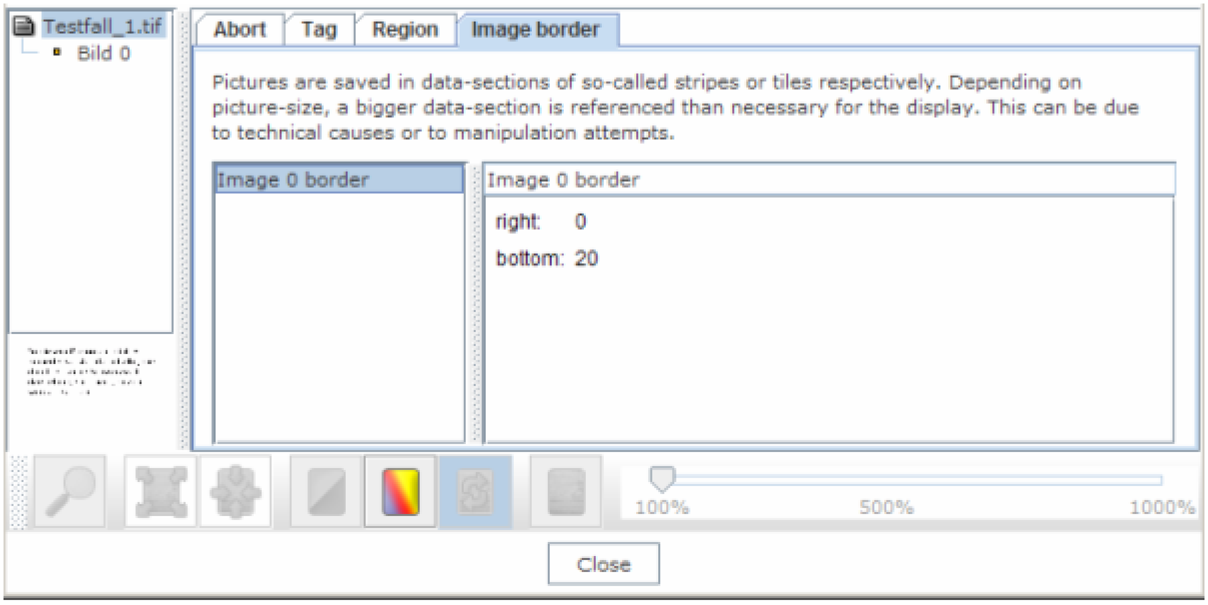
[✓ All] Relevant

Figure 28: Secure display of unreferenced regions

Picture borders

How borders occur in tiff documents: The picture area in a tiff file is either constructed of stripes or tiles. Every stripe or tile consists of a fixed number of pixels. If more pixels are contained, than can be partitioned into stripes or tiles with modulo 0, than the result is a border.

Files that have a border are potentially suspicious. For various reasons no assumption can be made about borders. Only if the warning dialog is not displayed and the borders are shown with the size "0 px" the assumption is valid that the tiff file is not manipulated. If a border is greater than "0 px" the tiff file maybe manipulated.



Pictures are saved in data-sections of so-called stripes or tiles respectively. Depending on picture-size, a bigger data-section is referenced than necessary for the display. This can be due to technical causes or to manipulation attempts.

Image 0 border

Image 0 border

right: 0

bottom: 20

Figure 29: Trusted tiff viewer with "Image border" tab

3 Validation Client Integration Edition

3.1 Introduction

The Signer function library (Signer Integration Edition) described here is used for integration into specific applications.

This Java function library is delivered as a web service made available via an HTTP-SOAP interface and as an API for use by third-party applications.

The SOAP interface is essentially a Signer application that can be called with various parameters by third-party applications.

“Silent installation” permits installation of Signer Integration Edition to be performed with no user interaction.

3.2 Signer Integration Edition - installation

The Signer Integration Edition is supplied as an executable installation file. You are guided step-by-step through various dialog windows in which you can configure a number of settings. After the initial dialog window, you can always return to the previous dialog if you need to make corrections by clicking the back button.

In the following dialog window, select the installation language. This language choice has no effect on the user interface of the Signer Integration Edition application.

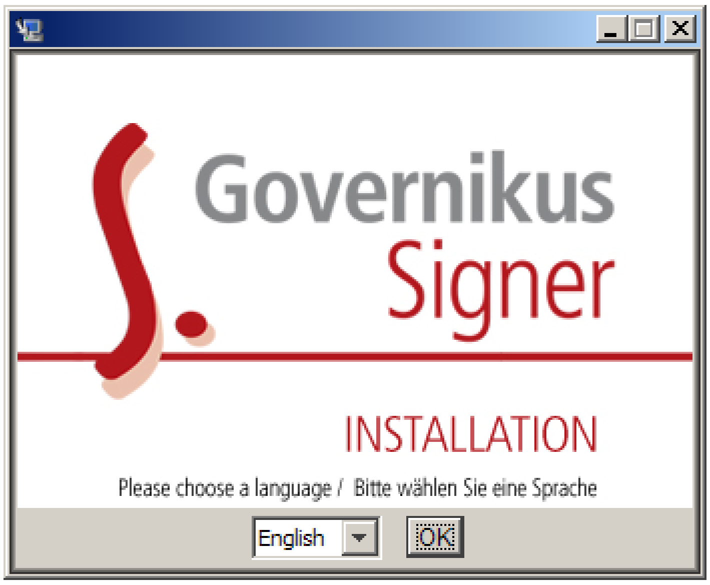
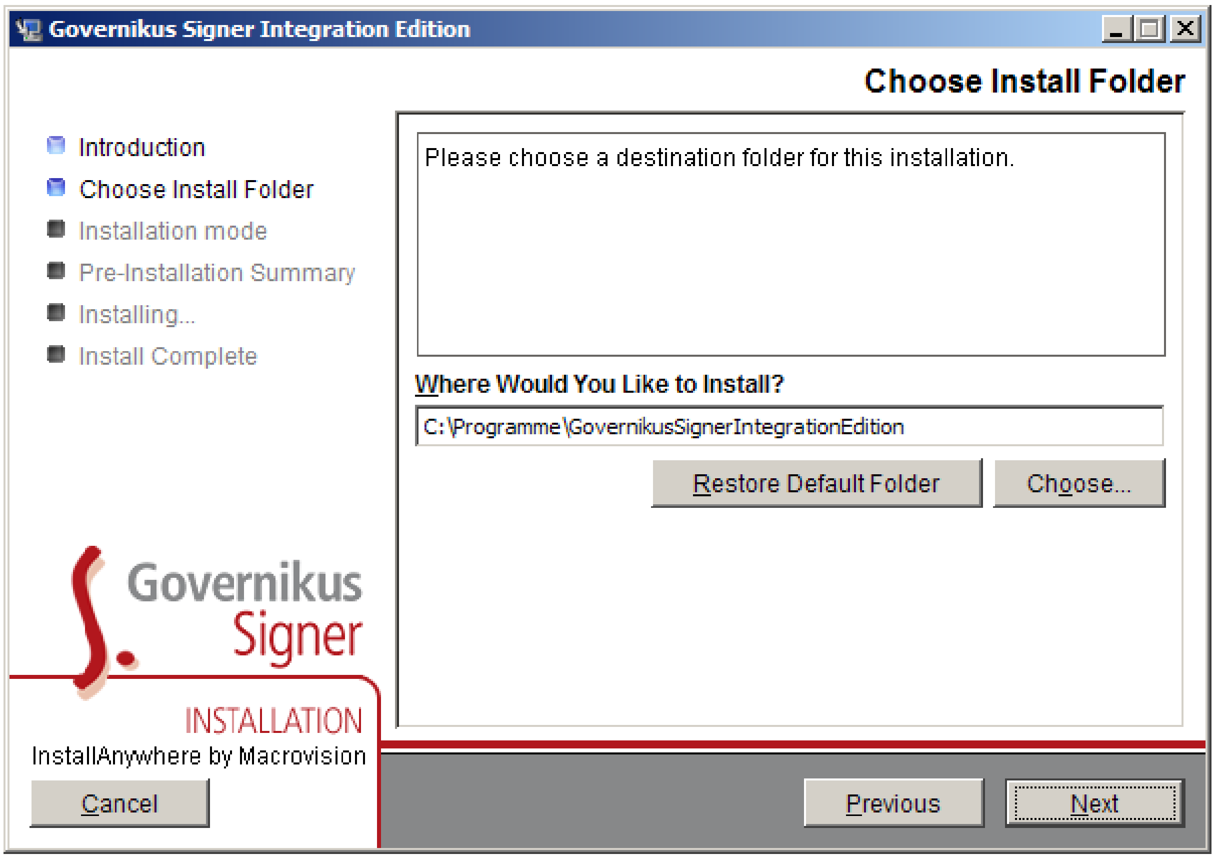


Figure 30: Language selection in the first dialog window

In this dialog window you can select an installation directory or accept the default settings.



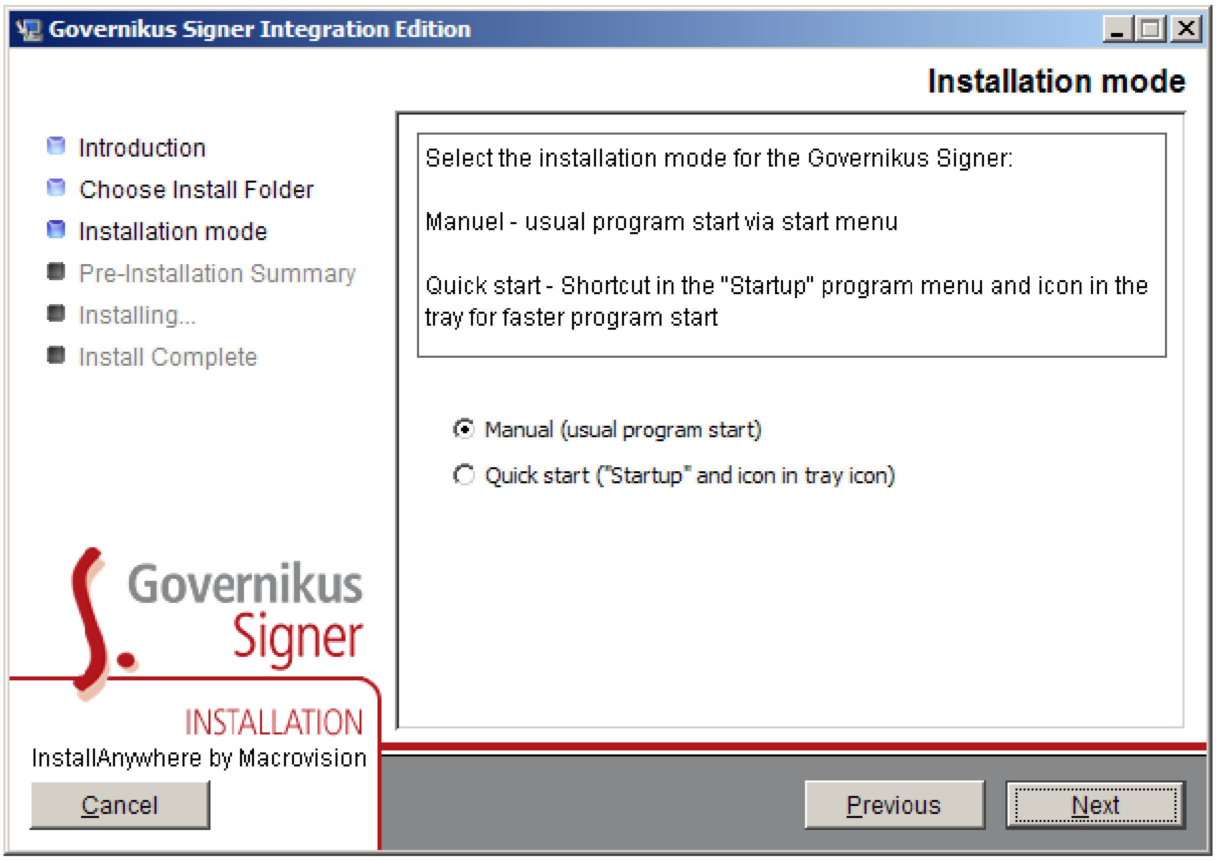
Please choose a destination folder for this installation.

Where Would You Like to Install?

C:\Programme\GovernikusSignerIntegrationEdition

Figure 31: Selecting the installation directory

In the next dialog window you can decide on the installation mode. You can choose between manual and quick start. Signer Integration Edition is exclusively run in the background. The user interface opens upon its first call via the interface. If you choose manual start Signer Integration Edition must be started via a start program (see chapter 3.2.4). If you choose the quick start option a respective start-up entry is created and Signer Integration Edition is automatically started on user login.



Select the installation mode for the Governikus Signer:

Manuel - usual program start via start menu

Quick start - Shortcut in the "Startup" program menu and icon in the tray for faster program start

[✓ Manual (usual program start)]
Quick start ("Startup" and icon in tray icon)

Figure 32: Selecting the installation mode

After this dialog window your installation selection is summarised on a further screen. Click on "Install" in the last dialog window. Click on “Finish” to complete the installation.

3.2.1 Configuration data

Signer always saves user configuration in a file. This file is always called bos_signer_V2.xml and is saved in the user directory %USERPROFILE% (Windows) or $HOME (Linux).

The location of the configuration file can be changed using the environment variable SignerConfigFolder:

Command: `set SignerConfigFolder=C:\Program Files\GovernikusSigner`

3.2.2 Signer - silent installation

The silent installation lets you install the Signer Integration Edition without prompting installation dialogs. However, precondition is a silent installation file that contains the necessary installation settings.

Creating a silent installation file

A command line call of the installation file with parameter “-r” launches the Signer Installation Edition installation routine and creates a silent installation file automatically. Example:

Command: `GovSigner_<version>.exe -r "c:\silent.properties"`

Note: In the above command line please replace <version> by the proper version number of your Signer Integration Edition

Editing the silent installation file

If the installation is to be user-specific the silent.properties file has to be edited. To save the links for the logged-in user under their "Start menu/All Programs", the USER_SHORTCUTS variable must be edited as follows:

```
USER_SHORTCUTS=$WIN_PROGRAMS_MENU$\\Governikus_Signer
```

Further changes can also be found in chapters about variables and the automatic installer of the Installanywhere user manual. The manual can be viewed here:

http://www.flexerasoftware.com/webdocuments/PDF/userguide_IA2008_en.pdf

Running silent installation

Use the following command line call to install Signer Integration Edition in silent mode. Example:

Command: `GovSigner_<version>.exe -i silent -f "c:\silent.properties"`

Note: In the above command line please replace <version> by the proper version number of your Signer Integration Edition

When Signer Integration Edition has been installed in silent mode, an installation log is written, which is saved as Governikus_Signer_InstallLog.log in the USER_INSTALL_DIR. The directory in which the application has been installed is given first in the installation log, followed by a summary, structured as shown below:

```
Summary
---------------
Installation: Successful.
117 Successful
0 Warnings
0 Errors, non-fatal
0 Errors, fatal
```

This information can be used to check that error-free installation of the application has been performed.

3.2.3 Signer in a terminal server environment

If Signer Integration Edition is used in a terminal server environment, i.e. where multiple users work on the same server system simultaneously, the following must be taken into account:

Note: If Signer Integration Edition is used in a Windows Server 2003 environment, the MultiClient.exe application must be launched first. This application manages the remote users' individual instances of Signer Integration Edition.

The MultiClient.exe application can be set up as a Windows service during installation.

MultiClient.exe

This application is only available for Windows Server 2003 and manages the remote users' individual instances of Signer Integration Edition. The application starts the servicePorts?wsdl web server on port 8086. If this port is unavailable or already in use, the environment variable SERVICEPORT can be used to define a different port. It should be noted that each instance of Signer Integration Edition requires two consecutive ports. Therefore the range above the specified ServicePort should be available.

Addressing the interface

To ensure that the application opens on the correct terminal the interface must always be addressed from the account of the user wishing to use Signer Integration Edition. In other words, the business process application from which Signer Integration Edition is to be called must also run under the respective user account.

3.2.4 Signer - first steps

Signer Integration Edition is exclusively started as background program and its availability is shown by an icon in the system tray. The user interface is displayed on calling the interface. On one computer only one instance of Signer Integration Edition can be started at the same time. Starting several instances is only possible in a terminal environment on Windows Server 2003 (see chapter 3.2.2). Starting and stopping Signer Integration Edition can be done in different ways.

3.2.5 Starting and stopping manually

For executing a call via the SOAP interface an instance of Signer Integration Edition must be started in advance. After successful installation two following executables are available in the installation directory:

startGovernikusSignerIntegrationEdition.exe

This executable starts Signer Integration Edition as a background application. As soon as the background application has been started, the SOAP interface is available and can be used.

Signer Integration Edition should always be called with ApplicationType hideAfterProcess via the interface. The user interface is then automatically closed after process execution. Thus the instance stays active in the background is not accidentally closed by the user.

More details/parameters relating to calling Signer Integration Edition can be found in the Developer Manual.

stopGovernikusSignerIntegrationEdition.exe

This call ends the instance of the Signer Integration Edition. If ending the process is done from within an application it should be checked by the web service getstatus whether Signer Integration Edition is in use (status working or processing). Please read the Signer Developer Guide for details about the web service getStatus. You can also terminate Signer Integration Edition via the context menu from the tray icon.

3.2.6 Quick start

If quick start was selected during installation a shortcut to the file startGovernikusSignerIntegrationEdition.exe was created in the start-up program group. Hence the file is executed on every user login.

3.2.7 Automatic start on call via interface

If Signer Integration Edition is started via the interface or command line and no instance of it is running the start is executed automatically in the background, analogous to starting it via startGovernikusSignerIntegrationEdition.exe. Stopping cannot be done automatically and hence must be done via the file:

`stopGovernikusSignerIntegrationEdition.exe`.

</ Administrator Guide>

< Developer Guide>

3.2.8 Signer licence files

General

Signer provides two licence files that may be used to configure the software.

- The product licence file is supplied with Signer and defines the general functional scope of the software.
- When Signer Integration Edition is called, the calling application may pass another licence file to the Signer. This licence file can only mask the product licence; no functions deactivated by the product licence file can be activated via this licence file.

3.2.9 Basic structure

The licence file is an XML document that for each subitem defines a licence type, and if applicable subitems.

Each licence type possesses three properties; "Editable", "Use in process" and "Display", which configure the software.

Of the eight possible combinations of these properties, only five combinations can be used in this context. The possible combinations are marked with an "x" in the table below.

Available licence types:

Licence type	**Editable**	**Use in process**	**Display**
`enable`	x	x	x
`hide`		x	
`hide_disable`			
`disable`			x
`notEditable`		x	x

Table 1: License types

3.2.10 Masking the two licence files

If a licence file is passed by a calling application, each licence type of the product licence is masked by the passed licence file.

During masking, each property ("Editable", "Use in process" and "Display") is masked individually.

- Example 1:

  - Product licence: hide (-, x, -)
  - Licence file: disable (-, -, x)
  - Result hide_disable (-, -, -)
- Example 2:
  - Product licence: disable (-, -, x)
  - Licence file: enable (x, x, x)
  - Result disable (-, -, x)
- Example 3:
  - Product licence: enable (x, x, x)
  - Licence file: notEditable (-, x, x)
  - Result: notEditable (-, x, x)

3.2.11 Interpretation of the Licence File

When importing the licence files, each subitem is masked with its node point.

This means it is impossible for a subitem of the licence type to be visible while its node point is invisible.

Example 1

Licence file:

```
<decrypt>
      enable
      <sourcefolder>
         notEditable
         <addfiles>enable</addfiles>
         <removefiles>disable</removefiles>
      </sourcefolder>
      <targetfolder>
         hide
         <foldertype>enable</foldertype>
         <unzip>enable</unzip>
      </targetfolder>
      <key>
         enable
         <software>enable</software>
      </key>
</decrypt>
```

Interpreted licence file:

```
<decrypt>
      enable
```

```
        <sourcefolder>
           notEditable
           <addfiles>notEditable</addfiles>
           <removefiles>disable</removefiles>
        </sourcefolder>
        <targetfolder>
           hide
           <foldertype>hide</foldertype>
           <unzip>hide</unzip>
        </targetfolder>
        <key>
           enable
           <software>enable</software>
        </key>
  </decrypt>
```

3.2.12 Content of the licence file

The licence file is structured as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<root displayName="Professional">
  <process>
    <decrypt>enable
      <key>enable
        <software>enable</software>
      </key>
      <targetfolder>enable
      <foldertype>enable</foldertype>
        <unzip>enable</unzip>
      </targetfolder>
      <sourcefolder>enable
        <removefiles>enable</removefiles>
        <addfiles>enable</addfiles>
      </sourcefolder>
    </decrypt>
    <encrypt>enable
      <encryptalgorithm>enable</encryptalgorithm>
      <key>enable
        <software>enable</software>
        <encryptmultiaddressee>enable</encryptmultiaddressee>
     </key>
```

```
   <targetfolder>enable
   <foldertype>enable</foldertype>
     <createzip>enable</createzip>
   </targetfolder>
   <sourcefolder>enable
     <removefiles>enable</removefiles>
     <addfiles>enable</addfiles>
  </sourcefolder>
</encrypt>
 <verify>enable
   <options>enable
     <onlineverify>enable</onlineverify>
   <targetfolder>enable</targetfolder>
     <verifyserver>enable</verifyserver>
   </options>
   <sourcefolder>enable
     <addfiles>enable</addfiles>
     <removefiles>enable</removefiles>
  </sourcefolder>
</verify>
 <sign>enable
   <options>enable
     <signpolicy>enable
       <overwrite>enable</overwrite>
       <seriell>enable</seriell>
       <parallel>hide_disable</parallel>
   </signpolicy>
     <signformat>enable
       <pkcs7detached>enable</pkcs7detached>
     <pdfinline>enable</pdfinline>
       <pkcs7enveloped>enable</pkcs7enveloped>
       <osci>hide_disable</osci>
     </signformat>
     <netsigner>enable</netsigner>
     <extended_pdfsignature>enable
       <reason>enable</reason>
   </extended_pdfsignature>
     <timestampserver>hide_disable</timestampserver>
   <timestamp>hide_disable</timestamp>
   <visualization>enable</visualization>
```

```
        <signaturealgorithm>enable</signaturealgorithm>
        </options>
        <sourcefolder>enable
        <addfiles>enable</addfiles>
          <removefiles>enable</removefiles>
        </sourcefolder>
       <key>enable
        <netsigner>hide_disable</netsigner>
          <software>enable</software>
        </key>
        <batchsignature>enable</batchsignature>
        <targetfolder>enable</targetfolder>
        <attributecertificate>hide</attributecertificate>
      </sign>
    </process>
    <general>
      <actions>
        <settings>enable
          <subsequentprocess>enable</subsequentprocess>
          <reset>enable</reset>
          <proxyserver>enable</proxyserver>
          <updateserver>hide_disable</updateserver>
          <splashscreen>enable</splashscreen>
          <protocol>enable</protocol>
        <language>enable</language>
        </settings>
        <initcard>enable</initcard>
        <export>enable</export>
        <import>enable</import>
        <magnifier>enable</magnifier>
        <reset>enable</reset>
        <update>enable</update>
      </actions>
      <integratedws>hide</integratedws>
      <menu>enable</menu>
    </general>
  </root>
```

3.2.13 Overview of functions

The various Signer Integration Edition calls are described individually below.

The signer?wsdl web service described here can be accessed on port 8088 on a stand-alone installation.

In a terminal server installation, the servicePorts?wsdl web service can be accessed on port 8086 through which the actual port of a user instance may be queried.

Default values

Many parameters can access default values (e.g. by specifying "useDefault" in a SOAP call). In this case the last value set is used. This means that the relevant value from the properties file ("bos_signer_V2.xml" under %userprofile%) is initially used when the application is started. If the user changes a setting via the GUI or if a setting is explicitly specified by a SOAP call, this change then becomes the future default value and is also passed across to the properties file.

3.2.14 Signer Integration Edition – web service

For calling Signer Integration Edition as a web service an HTTP URL is provided to which a SOAP message can be sent.

The figure below lists the WSDL with an example IP address that defines this web service. Two schema definitions are also listed in which complex passing and return parameters are defined.

```
<?xml version="1.0" encoding="UTF-8"?>
   <!--
      Published by JAX-WS RI at http://jax-ws.dev.java.net. RI's version is
      JAX-WS RI 2.1.4-hudson-208-.
   -->
   <!--
      Generated by JAX-WS RI at http://jax-ws.dev.java.net. RI's version is
      JAX-WS RI 2.1.4-hudson-208-.
   -->
<definitions
   xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-
wssecurity-utility-1.0.xsd"
   xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:tns="governikus"
   xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns="http://schemas.xmlsoap.org/wsdl/"
   targetNamespace="governikus" name="SignerIntegratedInstanceService">
   <types>
      <xsd:schema>
         <xsd:import namespace="http://jaxb.dev.java.net/array"
            schemaLocation="http://208.77.188.166:8088/signer?xsd=1" />
      </xsd:schema>
      <xsd:schema>
         <xsd:import namespace="governikus"
schemaLocation="http://208.77.188.166:8088/signer?xsd=2" />
      </xsd:schema>
   </types>
   <message name="verify">
      <part xmlns:ns1="http://jaxb.dev.java.net/array" name="inputFiles"
         type="ns1:stringArray" />
      <part name="checkCertificateStateOnline" type="tns:booleanTriState"
/>
      <part name="targetFolderType" type="tns:targetFolderType" />
      <part name="outputFolder" type="xsd:string" />
      <part name="applicationPolicy" type="tns:applicationPolicy" />
      <part name="processPolicy" type="tns:processPolicy" />
      <part name="licencelicenceXML" type="xsd:string" />
      <part name="language" type="xsd:string" />
   </message>
   <message name="verifyResponse">
      <part name="return" type="xsd:boolean" />
   </message>
   <message name="getVersion" />
   <message name="getVersionResponse">
      <part name="return" type="xsd:string" />
   </message>
   <message name="getStatus" />
   <message name="getStatusResponse">
      <part name="return" type="tns:frameworkStatus" />
   </message>
   <message name="sign">
      <part name="serialNumber" type="xsd:string" />
      <part xmlns:ns2="http://jaxb.dev.java.net/array" name="inputFiles"
         type="ns2:stringArray" />
      <part name="targetFolderType" type="tns:targetFolderType" />
      <part name="outputFolder" type="xsd:string" />
      <part name="signatureFormat" type="tns:signatureType" />
      <part name="enablePDFInline" type="tns:booleanTriState" />
      <part name="signaturePolicy" type="tns:signaturePolicy" />
      <part name="maxParallelSignature" type="xsd:int" />
      <part xmlns:ns3="http://jaxb.dev.java.net/array"
name="attributeCertificateFiles"
         type="ns3:stringArray" />
      <part name="applicationPolicy" type="tns:applicationPolicy" />
      <part name="processPolicy" type="tns:processPolicy" />
      <part name="licenceXML" type="xsd:string" />
      <part name="language" type="xsd:string" />
      <part name="signatureAlgorithm" type="tns:signatureAlgorithm" />
   </message>
   <message name="signResponse">
      <part name="return" type="xsd:boolean" />
   </message>
   <message name="encrypt">
      <part xmlns:ns4="http://jaxb.dev.java.net/array" name="serialNumber"
         type="ns4:stringArray" />
      <part xmlns:ns5="http://jaxb.dev.java.net/array"
name="certificateFiles"
         type="ns5:stringArray" />
      <part xmlns:ns6="http://jaxb.dev.java.net/array" name="inputFiles"
         type="ns6:stringArray" />
      <part name="targetFolderType" type="tns:targetFolderType" />
      <part name="outputFolder" type="xsd:string" />
      <part name="createZipArchivBeforeEncrypt" type="tns:booleanTriState"
/>
      <part name="applicationPolicy" type="tns:applicationPolicy" />
      <part name="processPolicy" type="tns:processPolicy" />
      <part name="licenceXML" type="xsd:string" />
      <part name="language" type="xsd:string" />
   </message>
   <message name="encryptResponse">
      <part name="return" type="xsd:boolean" />
   </message>
   <message name="decrypt">
      <part name="serialNumber" type="xsd:string" />
      <part xmlns:ns7="http://jaxb.dev.java.net/array" name="inputFiles"
         type="ns7:stringArray" />
      <part name="targetFolderType" type="tns:targetFolderType" />
      <part name="outputFolder" type="xsd:string" />
      <part name="extractZipArchiv" type="tns:booleanTriState" />
      <part name="applicationPolicy" type="tns:applicationPolicy" />
      <part name="processPolicy" type="tns:processPolicy" />
      <part name="licenceXML" type="xsd:string" />
      <part name="language" type="xsd:string" />
   </message>
   <message name="decryptResponse">
      <part name="return" type="xsd:boolean" />
```

```
    </message>
    <message name="checkCard">
       <part name="serialNumber" type="xsd:string" />
    </message>
    <message name="checkCardResponse">
       <part name="return" type="xsd:boolean" />
    </message>
    <message name="getAvailableCardReader" />
    <message name="getAvailableCardReaderResponse">
       <part name="return" type="tns:cardReaderArray" />
    </message>
    <message name="reinitCards" />
    <message name="reinitCardsResponse">
       <part name="return" type="xsd:boolean" />
    </message>
    <portType name="SignerIntegratedInstance">
       <operation name="verify"
          parameterOrder="inputFiles checkCertificateStateOnline
targetFolderType outputFolder applicationPolicy processPolicy licenceXML
language">
          <input message="tns:verify" />
          <output message="tns:verifyResponse" />
       </operation>
       <operation name="getVersion">
          <input message="tns:getVersion" />
          <output message="tns:getVersionResponse" />
       </operation>
       <operation name="getStatus">
          <input message="tns:getStatus" />
          <output message="tns:getStatusResponse" />
       </operation>
       <operation name="sign"
          parameterOrder="serialNumber inputFiles targetFolderType
outputFolder signatureFormat enablePDFInline signaturePolicy
maxParallelSignature attributeCertificateFiles applicationPolicy
processPolicy licenceXML language signatureAlgorithm">
          <input message="tns:sign" />
          <output message="tns:signResponse" />
       </operation>
       <operation name="encrypt"
```

```
         parameterOrder="serialNumber certificateFiles inputFiles
targetFolderType outputFolder createZipArchivBeforeEncrypt
applicationPolicy processPolicy licenceXML language">
         <input message="tns:encrypt" />
         <output message="tns:encryptResponse" />
      </operation>
      <operation name="decrypt"
         parameterOrder="serialNumber inputFiles targetFolderType
outputFolder extractZipArchiv applicationPolicy processPolicy licenceXML
language">
         <input message="tns:decrypt" />
         <output message="tns:decryptResponse" />
      </operation>
      <operation name="checkCard">
         <input message="tns:checkCard" />
         <output message="tns:checkCardResponse" />
      </operation>
      <operation name="getAvailableCardReader">
         <input message="tns:getAvailableCardReader" />
         <output message="tns:getAvailableCardReaderResponse" />
      </operation>
      <operation name="reinitCards">
         <input message="tns:reinitCards" />
         <output message="tns:reinitCardsResponse" />
      </operation>
   </portType>
   <binding name="SignerIntegratedInstancePortBinding"
type="tns:SignerIntegratedInstance">
      <soap:binding transport="http://schemas.xmlsoap.org/soap/http"
         style="rpc" />
      <operation name="verify">
         <soap:operation soapAction="" />
         <input>
            <soap:body use="literal" namespace="governikus" />
         </input>
         <output>
            <soap:body use="literal" namespace="governikus" />
         </output>
      </operation>
      <operation name="getVersion">
         <soap:operation soapAction="" />
         <input>
            <soap:body use="literal" namespace="governikus" />
         </input>
         <output>
            <soap:body use="literal" namespace="governikus" />
         </output>
      </operation>
      <operation name="getStatus">
         <soap:operation soapAction="" />
         <input>
            <soap:body use="literal" namespace="governikus" />
         </input>
         <output>
            <soap:body use="literal" namespace="governikus" />
         </output>
      </operation>
      <operation name="sign">
         <soap:operation soapAction="" />
         <input>
            <soap:body use="literal" namespace="governikus" />
         </input>
         <output>
            <soap:body use="literal" namespace="governikus" />
         </output>
      </operation>
      <operation name="encrypt">
         <soap:operation soapAction="" />
         <input>
            <soap:body use="literal" namespace="governikus" />
         </input>
         <output>
            <soap:body use="literal" namespace="governikus" />
         </output>
      </operation>
      <operation name="decrypt">
         <soap:operation soapAction="" />
         <input>
            <soap:body use="literal" namespace="governikus" />
         </input>
         <output>
```

```
            <soap:body use="literal" namespace="governikus" />
         </output>
      </operation>
      <operation name="checkCard">
         <soap:operation soapAction="" />
         <input>
            <soap:body use="literal" namespace="governikus" />
         </input>
         <output>
            <soap:body use="literal" namespace="governikus" />
         </output>
      </operation>
      <operation name="getAvailableCardReader">
         <soap:operation soapAction="" />
         <input>
            <soap:body use="literal" namespace="governikus" />
         </input>
         <output>
            <soap:body use="literal" namespace="governikus" />
         </output>
      </operation>
      <operation name="reinitCards">
         <soap:operation soapAction="" />
         <input>
            <soap:body use="literal" namespace="governikus" />
         </input>
         <output>
            <soap:body use="literal" namespace="governikus" />
         </output>
      </operation>
   </binding>
   <service name="SignerIntegratedInstanceService">
      <port name="SignerIntegratedInstancePort"
binding="tns:SignerIntegratedInstancePortBinding">
         <soap:address location="http://208.77.188.166:8088/signer" />
      </port>
   </service>
</definitions>
```

WSDL 1: HTTP://208.77.188.166:8088/signer?wsdl

```
<?xml version="1.0" encoding="UTF-8"?>
```

```
   <!--
      Published by JAX-WS RI at HTTP://jax-ws.dev.java.net. RI's version is
      JAX-WS RI 2.1.4-hudson-208-.
   -->
<xs:schema xmlns:xs="HTTP://www.w3.org/2001/XMLSchema"
   version="1.0" targetNamespace="HTTP://jaxb.dev.java.net/array">
   <xs:complexType name="stringArray" final="#all">
      <xs:sequence>
         <xs:element name="item" type="xs:string" minOccurs="0"
            maxOccurs="unbounded" nillable="true" />
      </xs:sequence>
   </xs:complexType>
</xs:schema>
```

Schema: HTTP://208.77.188.166:8088/signer?xsd=1

```
<?xml version="1.0" encoding="UTF-8"?>
   <!--
      Published by JAX-WS RI at http://jax-ws.dev.java.net. RI's version is
      JAX-WS RI 2.1.4-hudson-208-.
   -->
<xs:schema xmlns:tns="governikus"
xmlns:xs="http://www.w3.org/2001/XMLSchema"
   version="1.0" targetNamespace="governikus">
   <xs:complexType name="cardReader">
      <xs:sequence>
         <xs:element name="cardID" type="xs:string" minOccurs="0" />
         <xs:element name="certificates" type="tns:certificate"
            nillable="true" minOccurs="0" maxOccurs="unbounded" />
      </xs:sequence>
   </xs:complexType>
   <xs:complexType name="certificate">
      <xs:sequence>
         <xs:element name="commonName" type="xs:string" minOccurs="0" />
         <xs:element name="encoded" type="xs:string" minOccurs="0" />
         <xs:element name="serialnumber" type="xs:string"
            minOccurs="0" />
      </xs:sequence>
   </xs:complexType>
   <xs:complexType name="frameworkStatus">
      <xs:sequence>
         <xs:element name="results" type="tns:result" nillable="true"
```

```
            minOccurs="0" maxOccurs="unbounded" />
         <xs:element name="state" type="tns:frameworkState"
            minOccurs="0" />
      </xs:sequence>
   </xs:complexType>
   <xs:complexType name="result">
      <xs:sequence>
         <xs:element name="detailMessage" type="xs:string"
            minOccurs="0" />
         <xs:element name="inputFileName" type="xs:string"
            minOccurs="0" />
         <xs:element name="resultFiles" type="xs:string" nillable="true"
            minOccurs="0" maxOccurs="unbounded" />
         <xs:element name="returnCode" type="xs:int" />
      </xs:sequence>
   </xs:complexType>
   <xs:simpleType name="targetFolderType">
      <xs:restriction base="xs:string">
         <xs:enumeration value="sameAsSourceFolder" />
         <xs:enumeration value="oneSpecial" />
         <xs:enumeration value="useDefault" />
      </xs:restriction>
   </xs:simpleType>
   <xs:simpleType name="booleanTriState">
      <xs:restriction base="xs:string">
         <xs:enumeration value="yes" />
         <xs:enumeration value="no" />
         <xs:enumeration value="useDefault" />
      </xs:restriction>
   </xs:simpleType>
   <xs:simpleType name="applicationPolicy">
      <xs:restriction base="xs:string">
         <xs:enumeration value="showBeyond" />
         <xs:enumeration value="hideAfterProcess" />
         <xs:enumeration value="useDefault" />
      </xs:restriction>
   </xs:simpleType>
   <xs:simpleType name="processPolicy">
      <xs:restriction base="xs:string">
         <xs:enumeration value="skippCompletedStep" />
```

```
            <xs:enumeration value="firstStep" />
            <xs:enumeration value="lastStep" />
            <xs:enumeration value="runProcess" />
            <xs:enumeration value="useDefault" />
        </xs:restriction>
    </xs:simpleType>
    <xs:simpleType name="signatureType">
        <xs:restriction base="xs:string">
            <xs:enumeration value="osci" />
            <xs:enumeration value="envelopedPKCS7" />
            <xs:enumeration value="detachedPKCS7" />
            <xs:enumeration value="rawXML" />
            <xs:enumeration value="useDefault" />
        </xs:restriction>
    </xs:simpleType>
    <xs:simpleType name="signaturePolicy">
        <xs:restriction base="xs:string">
            <xs:enumeration value="overwrite" />
            <xs:enumeration value="parallel" />
            <xs:enumeration value="seriell" />
            <xs:enumeration value="useDefault" />
        </xs:restriction>
    </xs:simpleType>
    <xs:simpleType name="signatureAlgorithm">
        <xs:restriction base="xs:string">
            <xs:enumeration value="rsa_sha1" />
            <xs:enumeration value="rsa_sha256" />
            <xs:enumeration value="rsa_ripemd160" />
            <xs:enumeration value="useHeighest" />
            <xs:enumeration value="useDefault" />
        </xs:restriction>
    </xs:simpleType>
    <xs:simpleType name="frameworkState">
        <xs:restriction base="xs:string">
            <xs:enumeration value="none" />
            <xs:enumeration value="initializing" />
            <xs:enumeration value="starting" />
            <xs:enumeration value="online" />
            <xs:enumeration value="working" />
            <xs:enumeration value="finished" />
            <xs:enumeration value="cancelled" />
            <xs:enumeration value="processing" />
         </xs:restriction>
      </xs:simpleType>
      <xs:complexType name="cardReaderArray" final="#all">
         <xs:sequence>
            <xs:element name="item" type="tns:cardReader" minOccurs="0"
               maxOccurs="unbounded" nillable="true" />
         </xs:sequence>
      </xs:complexType>
   </xs:schema>
```

Schema: HTTP:// 208.77.188.166:8088/signer?xsd=2

3.2.14.1 Calling the sign web service via SOAP

The sign web service can be used to provide one or more files with an advanced or qualified signature. Messages sent to the sign service must contain the following parameters:

Parameter	Description
`serialNumber:String`	Fully-qualified file name of the software signature key to be used (.p12 file) or serial number of the signature card to be used to create the signature (see 3.1.8). If no software signature key and no serial number have been given or no card with this serial number exists, the user must select the correct key before the signature is created. This parameter must not be passed as NULL. If no key is pre-selected, an empty string "" must be passed as the parameter. If `USE_DEFAULT` is passed as the parameter, Signer uses the last saved setting.
`inputFiles:StringArray`	Files and directories to be signed. Passed as a list with fully-qualified file names. If a directory is passed, all files (no sub-directories) are imported and added to the file list. This list may also contain files that have already been signed. In the case of PKCS#7 detached signed files, the list must also contain the original file. However, of the two files only the existing signature file is signed. This parameter must not be passed as NULL. An empty list must be passed instead. The files must be locally accessible on the target system.

Parameter	Description

`targetFolderType:TargetFolderType`	Specifies whether the results files should be saved in the same file folder as the files that are to be signed.
	Possible parameter values:
	`<xs:enumeration value="sameAsSourceFolder" />`
	`<xs:enumeration value="oneSpecial" />`
	`<xs:enumeration value="useDefault" />`

`outputFolder:String`	Specifies the location to which the created signature files are saved. Only evaluated if the `targetFolderType` parameter is defined as `oneSpecial`. This location must be locally accessible on the target system.

`signatureFormat:SignatureType`	Specifies the signature format to be used for creating the signature. Possible parameters are:
	`<xs:enumeration value="envelopedPKCS7"/>`
	`<xs:enumeration value="detachedPKCS7" />`
	`<xs:enumeration value="useDefault" />`

`signatureAlgorithm:SignatureAlgorithm`	Specifies the signature algorithm to be used for creating the signature. Possible parameters are:
	`<xs:enumeration value="rsa_sha1"/>`
	`<xs:enumeration value="rsa_sha256" />`
	`<xs:enumeration value="rsa_ripemd160"/>`
	`<xs:enumeration value="useHeighest" />`
	`<xs:enumeration value="useDefault" />`

`enablePDFInline:BooleanTriState`	Specifies whether a PDF inline signature should be created for PDF files irrespective of `signatureFormat`. `useDefault` takes the Signer value. `<xs:enumeration value="yes"/>` `<xs:enumeration value="no" />` `<xs:enumeration value="useDefault" />`

`signaturePolicy:SignaturePolicy`	Specifies whether the signature to be created should be applied in parallel or serial with any existing signature, or should overwrite it. Possible parameters are:
	`<xs:enumeration value="overwrite" />`
	`<xs:enumeration value="seriell" />`
	`<xs:enumeration value="useDefault" />`

`signatureAlgorithm:SignatureAlgorithm`	Specifies whether the signature to be created should be applied in parallel or serial with any existing signature, or should overwrite it. Possible parameters are:
	`<xs:enumeration value="overwrite" />`
	`<xs:enumeration value="seriell" />`
	`<xs:enumeration value="useDefault" />`

Parameter	Description
`maxParallelSignature:int`	This parameter is not currently evaluated.
`attributeCertificateFiles:StringArray`	List of attribute certificate files to be used for signature creation. This parameter must not be passed as `NULL`. If no key is pre-selected, an empty string "" must be passed as the parameter.

`applicationPolicy: ApplicationPolicy`	Specifies whether Signer Integration Edition should close or minimise the UI after the signature has been created `hideAfterProcess` or whether the user interface should remain open `showBeyond`.
	Possible parameters are:
	`<xs:enumeration value="showBeyond" />`
	`<xs:enumeration value="hideAfterProcess" />`
	`<xs:enumeration value="useDefault" />`

`processPolicy:ProcessPolicy`	Specifiers whether the GUI should skip all workflow steps, whose required entries were completely passed parameter (skippCompletedStep), or whether the workflow should always start with the firt (firstStep) or last step (lastStep). Possible entries are:
	`<xs:enumeration value="skippCompletedStep" />`
	`<xs:enumeration value="firstStep" />`
	`<xs:enumeration value="lastStep" />`
	`<xs:enumeration value="useDefault" />`

`licenceXML:String`	The content of a licence file can be passed to the application as a parameter. This licence file content can mask the product licence. Details are in Chapter 2. If no content is to be delivered, an empty string "" must be passed as the parameter.
`language:String`	The language setting to be used for the application once started. Possible parameters are: `de` or `en` If the parameter is not passed, the system language setting is taken as `DEFAULT`.

Table 2: 3.2.14.1 Calling the sign web service via SOAP

Note: The `SOAP response` tells the calling application whether the `SOAP request` has been successfully received. The return type contains no additional information on whether the request could be successfully carried out. For this the `getStatus` service must be called.

3.2.14.2 Calling the verify web service via SOAP

The verify web service is used to verify one or more signature files. A mathematical check (integrity check) and optionally an online check are performed on the certificates used for the signature (validation). Messages sent to the verify service must contain the following parameters:

Parameter	Description
`inputFiles:StringArray`	Files and directories to be verified. Passed as a list with fully-qualified file names. If a directory is passed, all files (no sub-directories) are imported and added to the file list. This parameter must not be passed as `NULL`. An empty list must be passed instead. The files must be locally accessible on the target system.

`checkCertificateStateOnline:BooleanTriState`	Specifies whether online verification should be performed. `useDefault` takes the Signer value.
	
	`<xs:enumeration value="yes"/>`
	
	`<xs:enumeration value="no" />`
	
	`<xs:enumeration value="useDefault" />`

`targetFolderType:TargetFolderType`	Specifies whether the results files should be saved in the same file folder as the files that are to be verified. Possible parameter values:
	
	`<xs:enumeration value="sameAsSourceFolder"/>`
	
	`<xs:enumeration value="oneSpecial" />`
	
	`<xs:enumeration value="useDefault" />`

`outputFolder:String`	Specifies the location to which the created check log files are saved. Only evaluated if the `targetFolderType` parameter is defined as `oneSpecial`. The location must be locally accessible on the target system.

Parameter	Description
`applicationPolicy: ApplicationPolicy`	Specifies whether Signer Integration Edition should close or minimise the UI after the signature has been created `hideAfterProcess` or whether the user interface should remain open `showBeyond`. Possible parameters are: `<xs:enumeration value="showBeyond" />` `<xs:enumeration value="hideAfterProcess" />` `<xs:enumeration value="useDefault" />`
`processPolicy:ProcessPolicy`	Specifies whether the user interface should skip all workflow steps for which the required settings have been completely passed as parameters `skipCompletedStep` or whether the workflow should always start with the first `firstStep` or last `lastStep` step. Possible parameters are: `<xs:enumeration value="skippCompletedStep" />` `<xs:enumeration value="firstStep" />` `<xs:enumeration value="lastStep" />` `<xs:enumeration value="useDefault" />`
`licenceXML:String`	The content of a licence file can be passed to the application as a parameter. This licence file content can mask the product licence. Details are in Chapter 2. If no content is to be delivered, an empty string "" must be passed as the parameter.
`language:String`	The language setting to be used for the application once started. Possible parameters are: `de` or `en` If the parameter is not passed, the system language setting is taken as `DEFAULT`.

Table 3: Calling the verify web service via SOAP

Note: The SOAP response tells the calling application whether the SOAP request has been successfully received. The return type contains no additional information on whether the request could be successfully carried out. For this the getStatus service must be called.

3.2.14.3 Calling the encrypt web service via SOAP

The encrypt web service is used to encrypt one or more files to the S/MIME standard. Optionally, the source files can first be compressed together into a ZIP file, which is then encrypted. Messages sent to the encrypt service must contain the following parameters:

Parameter	Description
`serialNumber:StringArray`	List of signature card (see 3.1.8) serial numbers to be used for encryption. If no serial number has been specified or no card with this serial number exists, the user must select the correct certificate before encryption. This parameter must not be passed as NULL. If no certificate is pre-selected, an empty list "" must be passed as the parameter. If USE_DEFAULT is set as the parameter in the list, Signer uses the last saved setting.
`certificateFiles:StringArray`	Fully-qualified file names of the certificates to be used for encryption. If no filename has been specified, the user must select the correct certificate before encryption. This parameter must not be passed as NULL. If no certificate is pre-selected, an empty list "" must be passed as the parameter. The files must be locally accessible on the target system.
`inputFiles:StringArray`	Files and directories to be encrypted. Passed as a list with fully-qualified file names. If a directory is passed, all files (no sub-directories) are imported and added to the file list. This parameter must not be passed as NULL. An empty list must be passed instead. The files must be locally accessible on the target system.

`targetFolderType:TargetFolderType`	Specifies whether the results files should be saved in the same file folder as the files that are to be encrypted. Possible parameter values:
	
	`<xs:enumeration value="sameAsSourceFolder" />`
	
	`<xs:enumeration value="oneSpecial" />`
	
	`<xs:enumeration value="useDefault" />`

`outputFolder:String`	Specifies the location to which the encrypted files are saved. Only evaluated if the targetFolderType parameter is defined as oneSpecial. The location must be locally accessible on the target system.

Parameter	Description
`createZipArchivBeforeEncrypt:BooleanTriState`	Specifies whether the files should be compressed together into a ZIP archive before encryption. useDefault takes the Signer value. <xs:enumeration value="yes"/> <xs:enumeration value="no" /> <xs:enumeration value="useDefault" />

`applicationPolicy: ApplicationPolicy`	Specifies whether Signer Integration Edition should close or minimise the UI after the signature has been created hideAfterProcess or whether the user interface should remain open (showBeyond).
	
	Possible parameters are:
	
	`<xs:enumeration value="showBeyond" />`
	
	`<xs:enumeration value="hideAfterProcess" />`
	
	`<xs:enumeration value="useDefault" />`

`processPolicy:ProcessPolicy`	Specifies whether the user interface should skip all workflow steps for which the required settings have been completely passed as parameters skipCompletedStep or whether the workflow should always start with the first firstStep or last lastStep step. The process can be started directly runProcess if all the parameters have been completely specified. Possible parameters are:
	
	`<xs:enumeration value="skippCompletedStep" />`
	
	`<xs:enumeration value="firstStep" />`
	
	`<xs:enumeration value="lastStep" />`
	
	`<xs:enumeration value="runProcess" />`
	
	`<xs:enumeration value="useDefault" />`

`licenceXML:String`	The content of a licence file can be passed to the application as a parameter. This licence file content can mask the product licence. Details are in Chapter 2. If no content is to be delivered, an empty string "" must be passed as the parameter.
`language:String`	The language setting to be used for the application once started. Possible parameters are: "de" or "en" If the parameter is not passed, the system language setting is taken as DEFAULT.

Table 4: Calling the encrypt web service via SOAP

Note: The SOAP response tells the calling application whether the SOAP request has been successfully received. The return type contains no additional information on whether the request could be successfully carried out. For this the getStatus service must be called.

3.2.14.4 Calling the decrypt web service via SOAP

The decrypt web service is used to decrypt one or more files to the S/MIME standard. Messages sent to the decrypt service must contain the following parameters:

Parameter	Description
`serialNumber:String`	Fully-qualified file name of the software signature key to be used (.p12 file) or serial number of the signature card (see 3.1.8) to be used for decryption. If no software signature key and no serial number have been specified or no card with this serial number exists, the user must select the correct certificate before decryption. This parameter must not be passed as `NULL`. If no key is pre-selected, an empty string "" must be passed as the parameter. If `USE_DEFAULT` is passed as the parameter, Signer uses the last saved setting.
`inputFiles:StringArray`	Files and directories to be decrypted. Passed as a list with fully-qualified file names. If a directory is passed, all files (no sub-directories) are imported and added to the file list. This parameter must not be passed as `NULL`. An empty list must be passed instead. The files must be locally accessible on the target system.

`targetFolderType:TargetFolderType`	Specifies whether the results files should be saved in the same file folder as the files to be decrypted.
	
	Possible parameter values:
	`<xs:enumeration value="sameAsSourceFolder" />`
	`<xs:enumeration value="oneSpecial" />`
	`<xs:enumeration value="useDefault" />`

`outputFolder:String`	Specifies the location to which the decrypted files are saved. Only evaluated if the `targetFolderType` parameter is defined as `oneSpecial`. The location must be locally accessible on the target system.

`extractZipEncrypt:BooleanTriState`	Specifies whether a ZIP archive should be unpacked after decryption. `useDefault` takes the Signer value.
	`<xs:enumeration value="yes"/>`
	`<xs:enumeration value="no" />`
	`<xs:enumeration value="useDefault" />`

Parameter	Description
`applicationPolicy: ApplicationPolicy`	Specifies whether Signer Integration Edition should close or minimise the UI after the signature has been created `hideAfterProcess` or whether the user interface should remain open `showBeyond`. Possible parameters are: `<xs:enumeration value="showBeyond" />` `<xs:enumeration value="hideAfterProcess" />` `<xs:enumeration value="useDefault" />`
`processPolicy:ProcessPolicy`	Specifies whether the user interface should skip all workflow steps for which the required settings have been completely passed as parameters `skipCompletedStep` or whether the workflow should always start with the first `firstStep` or last `lastStep` step. The process can be started directly `runProcess` if all the parameters have been completely specified. Possible parameters are: `<xs:enumeration value="skippCompletedStep" />` `<xs:enumeration value="firstStep" />` `<xs:enumeration value="lastStep" />` `<xs:enumeration value="runProcess" />` `<xs:enumeration value="useDefault" />`
`licenceXML:String`	The content of a licence file can be passed to the application as a parameter. This licence file content can mask the product licence. Details are in Chapter 2. If no content is to be delivered, an empty string "" must be passed as the parameter.
`language:String`	The language setting to be used for the application once started. Possible parameters are: `"de"` or `"en"` If the parameter is not passed, the system language setting is taken as `DEFAULT`.

Table 5: Calling the decrypt web service via SOAP

Note: The SOAP response tells the calling application whether the SOAP request has been successfully received.

The return type contains no additional information on whether the request could be successfully carried out. For this the getStatus service must be called.

3.2.14.5 Calling the getStatus web service via SOAP

The getStatus web service is used to query the current processing status as well as the processing result. Messages sent to the getStatus service contain no parameters. The following values are returned to the calling application via the SOAP response:

Parameter	Description
`signerState`	Shows the overall status of the current process. Possible values are: • none

Parameter	Description
	• initializing (only when the `reinitCards` service is called) • starting • online • working • finished • cancelled • processing
`resultFiles`	A list of all files and their return code (i.e. separate return code for each file).

Table 6: Calling the getStatus web service via SOAP

3.2.14.6 Calling the reinitCards web service via SOAP

Using the reinitCards web service all card readers are re-imported and are then available to the signer.

The following values are returned to the calling application via the SOAP response:

Parameter	Description
boolean	Specifies whether reinitCards was successful.

Table 7: Calling the reinitCards web service via SOAP

3.2.14.7 Calling the checkCard web service via SOAP

The checkCard web service is used to check whether the desired signature card is available. Messages sent to the checkCard service contain the following parameters.

Parameter	Description
`serialNumber`	Specifies the serial number of the certificate to be checked.

Table 8: Calling the checkCard web service via SOAP

Note: The SOAP response tells the calling application whether a signature card linked to Signer Integration Edition contains a certificate with the specified serial number.

3.2.14.8 Calling the getAvailableCardReader web service via SOAP

The getAvailableCardReader web service is used to query a list of cards in available card readers. The following values are returned to the calling application via the SOAP response:

Parameter	Description
`CardReader`	`cardId` – ID of the card in the card reader
	`certificates` – List of certificates (Base64-encoded), their owner and their serial numbers

Table 9: Calling the getAvailableCardReader web service via SOAP

3.2.15 servicePorts Web Service

To call as a web service an HTTP URL is provided to which a SOAP message can be sent.

The servicePorts?wsdl web service is accessible by default on port 8086. Should this port be unavailable or already in use, the environment variable "SERVICEPORT" can be used to define a different port.

The figure below lists the WSDL with an example IP address that defines this web service.

```
<?xml version='1.0' encoding='UTF-8'?>
   <!--
      Published by JAX-WS RI at HTTP://jax-ws.dev.java.net. RI's version is
      JAX-WS RI 2.1.4-hudson-208-.
   -->
   <!--
      Generated by JAX-WS RI at HTTP://jax-ws.dev.java.net. RI's version is
      JAX-WS RI 2.1.4-hudson-208-.
   -->
<definitions
   xmlns:wsu="HTTP://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-
wssecurity-utility-1.0.xsd"
   xmlns:soap="HTTP://schemas.xmlsoap.org/wsdl/soap/"
xmlns:tns="governikus.service"
   xmlns:xsd="HTTP://www.w3.org/2001/XMLSchema"
xmlns="HTTP://schemas.xmlsoap.org/wsdl/"
   targetNamespace="governikus.service" name="MultiClientService">
   <types>
      <xsd:schema>
         <xsd:import namespace="governikus.service"
            schemaLocation="HTTP://208.77.188.166:8086/servicePorts?xsd=1"
/>
      </xsd:schema>
   </types>
   <message name="getCurrentPort">
      <part name="application" type="tns:application" />
      <part name="username" type="xsd:string" />
      <part name="clientName" type="xsd:string" />
      <part name="sessionName" type="xsd:string" />
   </message>
   <message name="getCurrentPortResponse">
      <part name="return" type="xsd:int" />
   </message>
   <portType name="MultiClient">
      <operation name="getCurrentPort"
```

```
            parameterOrder="application username clientName sessionName">
            <input message="tns:getCurrentPort" />
            <output message="tns:getCurrentPortResponse" />
        </operation>
    </portType>
    <binding name="MultiClientPortBinding" type="tns:MultiClient">
        <soap:binding transport="HTTP://schemas.xmlsoap.org/soap/HTTP"
            style="rpc" />
        <operation name="getCurrentPort">
            <soap:operation soapAction="" />
            <input>
                <soap:body use="literal" namespace="governikus.service" />
            </input>
            <output>
                <soap:body use="literal" namespace="governikus.service" />
            </output>
        </operation>
    </binding>
    <service name="MultiClientService">
        <port name="MultiClientPort" binding="tns:MultiClientPortBinding">
            <soap:address location="HTTP://208.77.188.166:8086/servicePorts" />
        </port>
    </service>
</definitions>
```

Schema: HTTP://208.77.188.166:8086/servicePorts?wsdl

```
<?xml version='1.0' encoding='UTF-8'?>
    <!--
        Published by JAX-WS RI at HTTP://jax-ws.dev.java.net. RI's version is
        JAX-WS RI 2.1.4-hudson-208-.
    -->
<xs:schema xmlns:xs="HTTP://www.w3.org/2001/XMLSchema"
    version="1.0" targetNamespace="governikus.service">
    <xs:simpleType name="application">
        <xs:restriction base="xs:string">
            <xs:enumeration value="GovernikusIntegratedSigner" />
            <xs:enumeration value="GovernikusSigner" />
        </xs:restriction>
    </xs:simpleType>
</xs:schema>
```

Schema: HTTP:// 208.77.188.166:8086/servicePorts?xsd=1

3.2.15.1 Calling the `getCurrentPort` web service via SOAP

The getCurrentPort web service is used to query the current port for a particular instance of Signer Integration Edition. Messages sent to the getCurrentPort service must contain the following parameters:

Parameter	Description
`application:Application`	`GovernikusIntegratedSigner` must always be passed as the application.
`username:String`	Content of the `USERNAME` environment variable. This contains the name of the remote user.
`clientName:String`	Content of the `CLIENTNAME` environment variable. This contains the computer name of the remote client.
`sessionName:String`	Content of the `SESSIONNAME` environment variable. This contains the `SessionID` of the remote session.

Table 10: Calling the `getCurrentPort` web service via SOAP

The following values are returned to the calling application via the `SOAP response`:

Parameter	Description
`Int`	Returns the port number of the queried instance of Signer Integration Edition.

Table 11: Return values

3.2.16 Return codes, error handling

The table below contains all the return codes in the product. This also includes some return codes that may be irrelevant to specific projects (e.g. if only software certificates are used).

If multiple errors arise, the application returns the return code with the highest ErrorLevel. If there are multiple return codes with the same ErrorLevel, the first code created is returned. This may mean that the return code KEY_NOT_VALID_YET is returned but the file has still been successfully saved as signed.

Return code	Value	Meaning	Error Level
0	`OK`	File(s) has/have been successfully signed.	1
1	`ABORTED`	Operation was cancelled.	4
2	`RESET`	User has selected "Reset reader configuration", application must be called again.	4
10	`MISSING_PARAM`	Missing parameter	4
11	`ILLEGAL_PARAM`	Parameter error	4
20	`FILE_NOT_FOUND`	Input file or directory unavailable	4
22	`SOURCE_DIR_EMPTY`	(Source) directory is empty.	4

Return code	Value	Meaning	Error Level
24	CONNECTION_ERROR	Error creating the connection to the server.	2
26	CANNOT_WRITE_TO_FOLDER	Cannot write to the target directory.	3
27	CANNOT_UNZIP	The ZIP archive cannot be unpacked.	3
31	CANNOT_OPEN_SOURCE	Error opening source file(s)	4
33	CANNOT_CLOSE_INPUT	Input file cannot be closed.	2
34	CANNOT_CLOSE_OUTPUT	Output file cannot be closed.	2
35	NO_ONLINE_VERIFY	Online verification was deactivated.	1
36	NO_CONTENT	No content data found for the verification of a PKCS#7 detached file.	4
38	CANNOT_SAVE_PROTOCOL	Verification log could not be saved.	4
40	KEY_NOT_FOUND	Signature key (software certificate) not available.	4
41	KEY_EXPIRED	Validity of signature key has expired.	3
42	KEY_NOT_VALID_YET	Validity of signature key has not yet commenced.	3
43	CANNOT_LOAD_KEYSTORE	Error loading signature key.	4
44	NO_KEYSTORE_FILE	Incorrect keystore format (software certificate). The specified file could not be interpreted as PKCS#12.	4
45	CARD_NOT_INITIALIZED	Signature card not yet activated (transport PIN is still assigned).	4
46	CARD_REMOVED	Signature card has been removed.	4
50	INVALID_PIN	Invalid PIN entry	4
51	TOO_MANY_WRONG_PINS	Number of incorrect PIN entries exceeded.	4
52	UNKNOWN_SIG_TYPE	Signature format not supported	4
53	UNKNOWN_SIG_ALGORITHM	Signature algorithm not supported	4
54	PIN_INPUT_CANCEL	PIN entry cancelled	4
60	CERTIFICATE_STATUS_UNKNOWN	Unknown certificate status	3
61	CERTIFICATE_INDETERMINATE_STATUS	INDETERMINATE certificate	2
62	CERTIFICATE_NOT_QUALIFIED	Not a qualified signature	2
65	CERTIFICATE_INVALID_STATUS_VALIDITYINTERVAL	INVALID certificate outside validity period	3
66	CERTIFICATE_INVALID_STATU	INVALID certificate RevocationStatus	3

Return code	Value	Meaning	Error Level
	S_REVOCATIONSTATUS		
67	CERTIFICATE_INVALID_STATUS_ISSUERTRUST	INVALID certificate IssuerTrust	3
68	CERTIFICATE_INVALID_STATUS_SIGNATURE	INVALID certificate Signature	2
70	SERVER_SIGNATURE_INVALID	The verification server response has been changed or an incorrect server certificate has been configured.	1
71	SERVER_SIGNATURE_NOT_FOUND	No signature found in the verification server response.	1
72	SERVER_CERTIFICATE_NOT_FOUND	No server certificate found in the configuration.	1
73	VERIFYTIME_MODIFIED	The verification time has been changed.	1
75	VERIFICATIONSERVER_ERROR	The verification server returned the following errors:	1
76	NETSIGNER_SHA1_CHECK_ERROR	The NetSigner certificate has been changed.	1
80	NO_PDF_FILE	"PDF Inline" signature format requested, file to be signed is not a PDF file	3
81	UNSUPPORTED_PDF_DIALECT	"PDF Inline" signature format requested, the PDF format of the document to be signed is not supported	3
85	CONTENT_MANIPULATED	File content has been manipulated.	
86	TIMESTAMP_RESPONSE_MANIPULATED	Timestamp has been manipulated.	
87	NETSIGNER_RESPONSE_MANIPULATED	NetSigner response has been manipulated.	
99	UNKNOWN	Unspecified processing error.	3

Table 12: Return codes, error handling

3.2.17 Examples

Signer Integration edition (Sign, Verify, Encrypt and Decrypt) can be called via an HTTP SOAP.

3.2.17.1 Sign

SOAP call:

This example is intended to show how the SOAP interface in Signer can be used to sign files.

```
<?xml version='1.0' encoding='UTF-8'?>
<S:Envelope xmlns:S="HTTP://schemas.xmlsoap.org/soap/envelope/">
   <S:Body>
      <ns2:sign xmlns:ns2="governikus">
```

```
         <serialNumber>./resources/GovernikusSigner.p12</serialNumber>
         <inputFiles>
            <item>File.txt</item>
         </inputFiles>
         <targetFolderType>oneSpecial</targetFolderType>
         <outputFolder>./test/dest/</outputFolder>
         <signatureFormat>detachedPKCS7</signatureFormat>
         <enablePDFInline>yes</enablePDFInline>
         <signaturePolicy>overwrite</signaturePolicy>
         <maxParallelSignautre>11</maxParallelSignautre>
         <attributeCertificateFiles />
         <applicationPolicy>hideAfterProcess</applicationPolicy>
         <processPolicy>skippCompletedStep</processPolicy>
         <licenceXML></licenceXML>
          <language>det</language>
      </ns2:sign>
   </S:Body>
</S:Envelope>
```

Example SOAP call (sign)

```
<?xml version='1.0' encoding='UTF-8'?>
<S:Envelope xmlns:S="HTTP://schemas.xmlsoap.org/soap/envelope/">
   <S:Body>
      <ns2:getStatus xmlns:ns2="governikus" />
   </S:Body>
</S:Envelope>
```

Example SOAP call (query status)

```
<?xml version='1.0' encoding='UTF-8'?>
<S:Envelope xmlns:S="HTTP://schemas.xmlsoap.org/soap/envelope/">
   <S:Body>
      30b
      <ns2:getStatusResponse xmlns:ns2="governikus">
         <return>
            <results>
               <detailMessage></detailMessage>
               <inputFileName>
                  /home/bos/GovernikusSigner/testmatrix/data/Encrypt/pdf.pdf
               </inputFileName>
               <returnCode>-1</returnCode>
            </results>
            <results>
               <detailMessage></detailMessage>
               <inputFileName>
                  /home/bos/GovernikusSigner/testmatrix/data/Encrypt/doc.doc
               </inputFileName>
               <returnCode>-1</returnCode>
            </results>
            <results>
               <detailMessage></detailMessage>
               <inputFileName>
   /home/bos/GovernikusSigner/testmatrix/data/Encrypt/encrypt.rar
               </inputFileName>
               <returnCode>-1</returnCode>
            </results>
            <state>online</state>
         </return>
      </ns2:getStatusResponse>
   </S:Body>
</S:Envelope>
```

Example SOAP response (`query status`)

3.2.17.2 Verify

SOAP call:

This example is intended to show how the SOAP interface in Signer can be used to verify files.

```
<?xml version='1.0' encoding='UTF-8'?>
<S:Envelope xmlns:S="HTTP://schemas.xmlsoap.org/soap/envelope/">
   <S:Body>
      <ns2:verify xmlns:ns2="governikus">
         <inputFiles>
            <item>File.pdf</item>
         </inputFiles>
       <checkCertificateStateOnline>yes</checkCertificateStateOnline>
         <targetFolderType>oneSpecial</targetFolderType>
         <outputFolder>./test/dest/</outputFolder>
         <applicationPolicy>hideAfterProcess
         </applicationPolicy>
         <processPolicy>skippCompletedStep</processPolicy>
         <licenceXML></licenceXML>
          <language>de</language>
      </ns2:verify>
   </S:Body>
</S:Envelope>
```

Example SOAP call (`decrypt`)

3.2.17.3 Encrypt

SOAP call:

This example is intended to show how the SOAP interface in Signer can be used to encrypt files.

```
<?xml version='1.0' encoding='UTF-8'?>
<S:Envelope xmlns:S="HTTP://schemas.xmlsoap.org/soap/envelope/">
   <S:Body>
      <ns2:encrypt xmlns:ns2="governikus">
         <serialNumber>
            <item>131597913404863079046787837746758669448</item>
         </serialNumber>
         <certificateFiles />
         <inputFiles>
            <item>D:\testmatrix\data\Encrypt\pdf.pdf</item>
            <item>D:\testmatrix\data\Encrypt\doc.doc</item>
            <item>D:\test\testmatrix\data\Encrypt\encrypt.rar</item>
         </inputFiles>
         <targetFolderType>oneSpecial</targetFolderType>
         <outputFolder>D:\testmatrix\dest</outputFolder>
         <createZipArchivBeforeEncrypt>yes</createZipArchivBeforeEncrypt>
         <applicationPolicy>showBeyond</applicationPolicy>
         <processPolicy>firstStep</processPolicy>
         <licenceXML></licenceXML>
          <language>de</language>
      </ns2:encrypt>
   </S:Body>
</S:Envelope>
```

Example SOAP call (`decrypt`)

3.2.17.4 Decrypt

SOAP call:

This example is intended to show how the SOAP interface in Signer can be used to decrypt files.

```
<?xml version='1.0' encoding='UTF-8'?>
<S:Envelope xmlns:S="HTTP://schemas.xmlsoap.org/soap/envelope/">
   <S:Body>
      <ns2:decrypt xmlns:ns2="governikus">
         <serialNumber>799150929</serialNumber>
```

```
            <inputFiles>
               <item>D:\testmatrix\data\Decrypt\Decrypt_1.zip.p7m</item>
            </inputFiles>
            <targetFolderType>oneSpecial</targetFolderType>
            <outputFolder>D:\testmatrix\dest</outputFolder>
            <extractZipArchiv>yes</extractZipArchiv>
            <applicationPolicy>showBeyond</applicationPolicy>
            <processPolicy>firstStep</processPolicy>
            <licenceXML></licenceXML>
             <language>de</language>
         </ns2:decrypt>
      </S:Body>
   </S:Envelope>
```

Example SOAP call (`encrypt`)

3.2.17.5 CheckCard

SOAP call:

This example is intended to show how the "checkCard" SOAP interface can be used in Signer.

```
<?xml version='1.0' encoding='UTF-8'?>
<S:Envelope xmlns:S="HTTP://schemas.xmlsoap.org/soap/envelope/">
   <S:Body>
      <ns2:checkCard xmlns:ns2="governikus">
         <serialNumber>89490173300004062580</serialNumber>
      </ns2:checkCard>
   </S:Body>
</S:Envelope>
```

Example SOAP call (`checkCard`)

```
<?xml version='1.0' encoding='UTF-8'?>
<S:Envelope xmlns:S="HTTP://schemas.xmlsoap.org/soap/envelope/">
   <S:Body>
      <ns2:checkCardResponse xmlns:ns2="governikus">
         <return>true</return>
      </ns2:checkCardResponse>
   </S:Body>
</S:Envelope>
```

Example SOAP response (`checkCard`)

3.2.17.6 ReinitCards

SOAP call:

This example is intended to show how the "reinitCards" SOAP interface can be used in Signer.

```
<?xml version='1.0' encoding='UTF-8'?>
<S:Envelope xmlns:S="HTTP://schemas.xmlsoap.org/soap/envelope/">
   <S:Body>
      <ns2:reinitCards xmlns:ns2="governikus" />
   </S:Body>
</S:Envelope>
```

Example SOAP call (`reinitCards`)

```
<?xml version='1.0' encoding='UTF-8'?>
<S:Envelope xmlns:S="HTTP://schemas.xmlsoap.org/soap/envelope/">
   <S:Body>
      <ns2:reinitCardsResponse xmlns:ns2="governikus">
         <return>true</return>
      </ns2:reinitCardsResponse>
   </S:Body>
</S:Envelope>
```

Example SOAP response (`reinitCards`)

3.2.17.7 GetAvailableCardReader

SOAP call:

This example is intended to show how the "getAvailableCardReader" SOAP interface can be used in Signer.

```
<?xml version='1.0' encoding='UTF-8'?>
<S:Envelope xmlns:S="HTTP://schemas.xmlsoap.org/soap/envelope/">
   <S:Body>
      <ns2:getAvailableCardReader xmlns:ns2="governikus" />
   </S:Body>
</S:Envelope>
```

Example SOAP call (`getAvailableCardReader`)

```
<?xml version='1.0' encoding='UTF-8'?>
<S:Envelope xmlns:S="HTTP://schemas.xmlsoap.org/soap/envelope/">
   <S:Body>
      <ns2:getAvailableCardReaderResponse
         xmlns:ns2="governikus">
         <return>
            <item>
               <cardID>89490173300004062580</cardID>
```

```
                <certificates>
                    <commonName>Pseudonym GovSigner :PN</commonName>
                    <encoded>
MIIF/TCCBOWgAwIBAgIEL64nijANBgkqhkiG9w0BAQsFADCBqzELMAkGA1UEBhMCREUxHDAaBgNV

    BAoME0RldXRzY2hlIFRlbGVrb20gQUcxJDAiBgNVBAsMG1QtVGVsZVNlYyBUZXN0IFRydXN0IENl

    bnRlcjElMCMGA1UEAwwcVC1UZWxlU2VjIFNpZ0cgVGVzdCBDQSAxNjpQTjEKMAgGA1UEBRMBMTEl

    MCMGA1UEQQwcVC1UZWxlU2VjIFNpZ0cgVGVzdCBDQSAxNjpQTjAeFw0wODAzMDUxNDAwNDZaFw0x

    MTAzMDUxNDAwNDZaMIGxMQswCQYDVQQGEwJERTEgMB4GA1UECgwXQnJlbWVuIE9ubGluZSBTZXJ2

    aWNlcyAxLjAsBgNVBAsMJVNtYXJ0Y2FyZCBhbmQgVHJ1c3RjZW50ZXIgSW50ZWdyYXRpb24xITAf

    BgNVBAMMGFBzZXVkb255bSBHb3ZTaWduZXIgIDpQTjEhMB8GA1UEQQwYUHNldWRvbnltIEdvdlNp

    Z25lciAgOlBOMQowCAYDVQQFEwExMIIBIzANBgkqhkiG9w0BAQEFAAOCARAAMIIBCwKCAQEAtuMV

    Y0odZIJVkJR82PZJM8rXaHIv1xqXw7Q/l+QXDnRzbHsgKGOipkCwQj9Vdy5AmXDBnL8eQGMZqYuF

    2+vZgO2Lk0RuwGJQSGTU/yDFAAXsmAkCMME7pVWmShJbvPepQeQiwRSZW0dKGoXrk7xb9x5GuK+w

    eQcNG1qoQxIm5q3PAomV7Iw7autavQoeYH35Ew6ZP3zC3aHzxWlOyQyOb6E3H1IOcr52tPWwQL8n

    KBoKEeOgaDEfV9JCK4SC+aYhHd8sT1o51VrTO30o4bLamTkA31b9vneBQVIdGCw19Gt3XVplX5MA

    DHsXvSu50IbCinY6ccTuaaXNoOngZboB4QIEQAAAgaOCAh4wggIaMB8GA1UdIwQYMBaAFAdLoEwT

    YJtz+X/Pn9G7NfQBK+iCMB0GA1UdDgQWBBRshDeP8uLKB6yRmcnmyhv8jVsARzAOBgNVHQ8BAf8E

    BAMCBkAwVAYDVR0gBE0wSzAHBgUrJAgBATBABgkCggYBCgECBQQwMzAxBggrBgEFBQcCARYlQ2Vy

    dGlmaWNhdGlvbiBQcmFjdGljZSBTdGF0ZW1lbnQgU2lnZzAbBgNVHREEFDASgRB0bkBib3MtYnJl

    bWVuLmRlMIH0BgNVHR8EgewwgekwgeagaqBohjVsZGFwOi8vcGtzbGRhcC50dHRjLmRlOjM4OS9v

    PURldXRzY2hlIFRlbGVrb20gQUcsYz1kZYYvaHR0cDovL3d3dy50dHRjLmRlL3RlbGVzZWMvc2Vy

    dmxldC9kb3dubG9hZF9jcmyieKR2MHQxCzAJBgNVBAYTAkRFMRwwGgYDVQQKFBNEZXV0c2NoZSBU

    ZWxla29tIEFHMRcwFQYDVQQLFA5ULVRlbGVTZWMgVGVzdDEuMAwGBwKCBgEKBxQTATEwHgYDVQQD

    FBdULVRlbGVTZWMgVGVzdCBESVIgODpQTjA0BggrBgEFBQcBAQQoMCYwJAYIKwYBBQUHMAGGGGh0

    dHA6Ly93d3cudHR0Yy5kZS9vY3NwcjAYBggrBgEFBQcBAwQMMAowCAYGBACORgEBMA4GBwKCBgEK

    DAAEAwEB/zANBgkqhkiG9w0BAQsFAAOCAQEAXXSkS4Y/38RMdjs57myCa7I/R73PhEeZjNx4l6ff

    rg8eTf3i6lyILM3y43UGnyzZsP5C+qUiC2rRnGxPlJaAIjdVoL52ZgzcatqxiUS27enkf9O0211e

    LH3690KwqrD77G1LyHIBmwrDzk13yb900qx1+/Ny6ci54pCYfuMapXWW9l46KVGhAD3k3EDUwqd+

    FcUXzksARs/2ZD7Xno7eW9Kv2SWtMk5XIv4KLkgJ7+Zuv8+0khDWJQupBQjeXn75OP1z0IN1/fcv

    afakuzELclO634LGoxj6iaxBkajXIBo/L3eXkqO7gVcqX7aSMm3eNC/vawgySjyeQbaehvMKzA==
</encoded>
                  <serialnumber>799942538</serialnumber>
                </certificates>
                <certificates>
                  <commonName>Pseudonym GovSigner :PN</commonName>
                  <encoded>
MIIF2zCCBMOgAwIBAgIEL64njDANBgkqhkiG9w0BAQUFADCBtjELMAkGA1UEBhMCREUxKzApBgNV

    BAoMIlQtU3lzdGVtcyBFbnRlcnByaXNlIFNlcnZpY2VzIEdtYkgxJDAiBgNVBAsMG1QtVGVsZVNl

    YyBUZXN0IFRydXN0IENlbnRlcjEjMCEGA1UEAwwaVC1TeXN0ZW1zIFBLUyBUZXN0IENBIDE6UE4x

    CjAIBgNVBAUTATExIzAhBgNVBEEMGlQtU3lzdGVtcyBQS1MgVGVzdCBDQSAxOlBOMB4XDTA4MDMw

    NTE0MDA1M1oXDTExMDMwNTE0MDA1M1owgbExCzAJBgNVBAYTAkRFMSAwHgYDVQQKDBdCcmVtZW4g

    T25saW5lIFNlcnZpY2VzIDEuMCwGA1UECwwlU21hcnRjYXJkIGFuZCBUcnVzdGNlbnRlciBJbnRl

    Z3JhdGlvbjEhMB8GA1UEAwwYUHNldWRvbnltIEdvdlNpZ25lciAgOlBOMSEwHwYDVQRBDBhQc2V1

    ZG9ueW0gR292U2lnbmVyICA6UE4xCjAIBgNVBAUTATEwggEjMA0GCSqGSIb3DQEBAQUAA4IBEAAw

    ggELAoIBAQCipLChVAus22hLwltAwSE0LaL7xUzJSDodzIXHcDxreqPIh5rKQAevb6dQEoV/hjNR

    1NUWZaZq0CMBeMei3nkJaPP5Q025kRp0BxOHQomHVgxloYHqaFouMPSAi2e252oHc6BmiwoIArR6

    MP4t9xpBufs2tMvlK+7RLvZCvht99lQwpZyuy7QhJEnsq+g5w6NELbHNofKtrAjWHYH/9jR2Uo0J

    BxZvjQ0wbb+809hn+etEuLyGtftriz0NkZ8jauLgTPr4d0+4Rda4YhD3tSRX8ll7TvZXTtlJo19d

    kKL/Tru4en80UeAK0O9b1QcvC1tqAE2HhqP6d9kytJ+cGS5ZAgRAAACBo4IB8TCCAe0wHwYDVR0j

    BBgwFoAUjU2S1u/LC+2QUx9H7nskc5rXgl4wUQYDVR0gBEowSDBGBgkCggYBCgECBQQwOTA3Bggr

    BgEFBQcCARYrQ2VydGlmaWNhdGlvbiBQcmFjdGljZSBTdGF0ZW1lbnQgbmljaHQgU2lnRzAdBgNV

    HQ4EFgQU9obIQZV76kIQ6pwBmwo2xvPBVrMwDgYDVR0PAQH/BAQDAgeAMBsGA1UdEQQUMBKBEHRu

    QGJvcy1icmVtZW4uZGUwgfQGA1UdHwSB7DCB6TCB5qBqoGiGNWxkYXA6Ly9wa3NsZGFwLnR0dGMu

    ZGU6Mzg5L289RGV1dHNjaGUgVGVsZWtvbSBBRyxjPWRlhi9odHRwOi8vd3d3LnR0dGMuZGUvdGVs

    ZXNlYy9zZXJ2bGV0L2Rvd25sb2FkX2NybKJ4pHYwdDELMAkGA1UEBhMCREUxHDAaBgNVBAoUE0Rl

    dXRzY2hlIFRlbGVrb20gQUcxFzAVBgNVBAsUDlQtVGVsZVNlYyBUZXN0MS4wDAYHAoIGAQoHFBMB

    MTAeBgNVBAMUF1QtVGVsZVNlYyBUZXN0IERJUiA4OlBOMDQGCCsGAQUFBwEBBCgwJjAkBggrBgEF

    BQcwAYYYaHR0cDovL3d3dy50dHRjLmRlL29jc3ByMA0GCSqGSIb3DQEBBQUAA4IBAQBLg5zXNoxF
                             2C4KTTS 1d3
                             LK8rp3iFiKrqJNjRmln+uyk09tSPhvh50bgA2327Xc1nb+ZDqdV4FG+4UcfyDRJ9/2qKy

    x3v5hlxyBeLopHOw++GnUDRewH2XOY8kH+ixsxWtWB8cJxhBKTFdu679cpXFN481OjB2JI82EXaB
```

```
   SUEDfNVjZiIUxObQhXobu8XX8gaXv7g7KsuURwroEK1xXMlYpWwJTEH9iQd0xUDnQG2fxs4cSXk0

   63T83BEcbG6MLeBeEDZqNMl6ktsBLxHjo4QsSPhq6AShYL+EetUOSejA4KUK2OHaoUenKodPSnSC
                                zUKdBzBfjknKxVm+vjUxsOCsqKIw
</encoded>
                 <serialnumber>799942540</serialnumber>
              </certificates>
           </item>
        </return>
     </ns2:getAvailableCardReaderResponse>
  </S:Body>
</S:Envelope>
```

Example SOAP response (getAvailableCardReader)

3.2.17.8 GetVersion

SOAP call:

This example is intended to show how the "getVersion" SOAP interface can be used in Signer.

```
<?xml version='1.0' encoding='UTF-8'?>
<S:Envelope xmlns:S="HTTP://schemas.xmlsoap.org/soap/envelope/">
  <S:Body>
     <ns2:getVersion xmlns:ns2="governikus" />
  </S:Body>
</S:Envelope>
```

Example SOAP call (getVersion)

```
<?xml version='1.0' encoding='UTF-8'?>
<S:Envelope xmlns:S="HTTP://schemas.xmlsoap.org/soap/envelope/">
  <S:Body>
     <ns2:getVersionResponse xmlns:ns2="governikus">
        <return>2.2.0.0</return>
     </ns2:getVersionResponse>
  </S:Body>
</S:Envelope>
```

Example SOAP response (getVersion)

3.3 Introduction SDK Guide

The Signer Integration Edition Software Development Kit (SDK) described here for connecting the Signer function library (Signer Integration Edition) is used for integration into specific applications.

This is a Java function library that forwards requests from third-party applications via the web services provided by the Signer Integration Edition through an HTTP SOAP interface.

The interface is essentially a Signer application that third-party applications are able to call with various parameters.

More information on Signer Integration Edition and the SOAP interface, as well as the structure of the licence file, can be found in the Governikus Integrated Signer Developer Manual PDF.

3.4 Signer Licence Files

General

Signer provides two licence files that may be used to configure the software.

- The product licence file is supplied with Signer and defines the general functional scope of the software.
- When Signer Integration Edition is called, the calling application may pass another licence file to the Signer. This licence file can only mask the product licence; no functions deactivated by the product licence file can be activated via this licence file.

Basic Structure

The basic structure and content of the licence file are described in detail in the bos Signer Developer Manual.

3.5 Overview of functions

The various Signer Integration Edition SDK calls are described individually below.

Default values

Many parameters can use default values. If a default value is described as useDefault, the last value set is used. This means that the relevant value from the properties file (bos_signer_V2.xml under %userprofile%) is initially used when the application is started. If the user changes a setting via the GUI or if a setting is explicitly specified by a SOAP call, this change then becomes the future default value and is also passed across to the properties file.

3.5.1 Signer Integration Edition – API / command line call

This is a Java function library that is made available in the form an API for use by third-party applications. We must here distinguish a local call from an external call made by an instance of Signer Integration Edition.

The functionalities of the call by an instance of Signer are described in the next chapters.

For the local Signer Integration Edition call, which may if required start a new instance, GovernikusIntegratedSigner.jar together with GovernikusSigner.jar and all its referenced function libraries must be integrated.

The functionality of the GovernikusIntegratedSigner.jar can be used for the external Signer Integration Edition call. However, it is essential to specify the URL of a Signer Integration Edition web service.

3.5.1.1 Functionality: Sign

For the local call an API is initialised, to which the following parameters can be passed.

- source (mandatory): Fully-qualified file name or directory with the files to be signed. The files must first be originated or provided by the third-party application.

- sigFormat (mandatory): Specifies the signature format as values ENVELOPED_PKCS_7|DETACHED_PKCS_7
- LOG_File (mandatory): File name (e.g. Sign.log) for the log file saved in the dest directory.
- dest (optional): Fully-qualified name of the directory in which the signed files should be saved. The signed file is saved in the dest directory. If DETACHED_PKCS_7 has been specified as the signature format by the calling application, the original file connected to the signature will also be copied to the dest directory.
- targetFolderType (optional): Specifies the TargetFolderType as values SAME_AS_SOURCE_FOLDER|ONE_SPECIAL|USE_DEFAULT.
  SAME_AS_SOURCE_FOLDER cannot be set if a dest has been specified. ONE_SPECIAL is then always used.
- keyStore (optional):Fully-qualified file name of the software signature key (.p12 file) to be used; if omitted, a signature card is used. If a serialnumber is passed in addition to the keyStore, only the serial number will be used. If USE_DEFAULT is passed as the parameter, Signer uses the last saved setting.

If "automatic" is passed as the parameter and a card reader with signature card is available, this signature card is selected automatically.

- serialnumber (optional): Here can be specified which card is to be used for signature. The serial number of the card must also be given. If a serialnumber is passed at the same time as a keyStore, only the serialnumber will be used. If USE_DEFAULT is passed as the parameter, Signer uses the last saved setting.

If "automatic" is passed as the parameter and a card reader with signature card is available, this signature card is selected automatically.

- sigPolicy (optional): Specifies the signature policy in the event that a file has already been signed, as values OVERWRITE|SERIAL
- webserviceURL (optional): The URL at which the Governikus Integrated Service is to be made available can be specified here. (e.g. http://208.77.188.166:8088/signer?wsdl)
- enablePdfInline (optional): This parameter (YES/NO/USE_DEFAULT) can optionally be specified on a call, to sign PDF files in the same format as all others. If this parameter is not specified the Signer setting is used. The parameter is not applicable to files that are not in PDF format.
- licenceFile (optional): The licence file to be used for masking can be specified here. The path to the file incl. file name or file URL must be passed for this.
- attributeCertificateFiles (optional): The fully-qualified attribute certificates can be specified here. IMPORTANT: Evaluation is not currently performed!
- maxParallelSignature (optional): The number of parallel signatures that can be applied can be specified here.
  IMPORTANT: Evaluation is not currently performed!
- applicationPolicy (optional): This can be used to specify whether the user interface should be closed or minimised after signature creation (HIDE_AFTER_PROCESS) or should remain open (SHOW_BEYOND). Specified in values: SHOW_BEYOND|HIDE_AFTER_PROCESS|USE_DEFAULT
- processPolicy (optional): Can be used to specify whether the user interface should skip all workflow steps for which the required settings have been completely passed as parameters (SKIPP_COMPLETED_STEP) or whether the workflow should always start with the first

(FIRST_STEP) or last (LAST_STEP) step. Specified in values: SKIPP_COMPLETED_STEP|FIRST_STEP|LAST_STEP|USE_DEFAULT

- signatureAlgorithm (optional): Which signature algorithm is to be used for signing can be specified here. USE_DEFAULT uses the last setting saved. USE_HIGHEST uses the highest possible signature algorithm. Specified in values: RSA_SHA_1|RSA_SHA_256|RSA_RIPEMD_160|USE_HEIGHEST|USE_DEFAULT
- maxGetWebserviceRetryCount (optional): The number of attempts to be made to connect to the web service can be specified here. An interval of 500 ms is left between each attempt to create a connection. If this parameter is not specified the number of attempts is set to 200.
- maxStartSignerRetryCount (optional): The number of attempts to be made to start Signer can be specified here. An interval of 500 ms is left between each attempt to create a connection. If this parameter is not specified the number of attempts is set to 100.
- language (optional): The language setting for the application can be passed here. Possible values are "de" and "en". The system language is used as the default.
- Help: Outputs all parameters and return codes with a corresponding description on the console.

Note: If only this parameter is passed to the application, this terminates after the console output with the return code (100) MISSING_PARAMETER.

- version: Outputs the version and build number on the console (see also note to help).

Attention: The directory containing the files to be signed and the destination directory must be specified in full by the calling application!

Additional conditions:

- All file operations (source and dest directory) must be locally accessible on the target system.

This API can be called using JAVA API or via the command line. If the call is made via API, the mandatory parameters must be passed to the constructor with the supplied SignParams class. Optional parameters can be passed to the SignParams class via setter methods.

If the call is made via the command line, all required parameters must be passed with the call. Call:

```
java –cp CLASSPATH de.bos_bremen.ecard.client.integrated.sign.Sign
-source VALUE [-source VALUE, …]
-sigFormat VALUE
-LOG_File VALUE
[-dest VALUE]
[-keyStore VALUE]
[-enablePdfInline VALUE]
[-serialnumber VALUE]
[-webserviceURL VALUE]
[-sigPolicy VALUE]
[-signatureAlgorithm VALUE]
[-licenceFile VALUE]
[-maxParallelSignature VALUE]
[-maxAttributeCertificateFiles VALUE]
[-targetFolderType VALUE]
[-applicationPolicy VALUE]
[-processPolicy VALUE]
```

```
[-maxGetWebserviceRetryCount VALUE]
[-maxStartSignerRetryCount VALUE]
[-language VALUE]
```

If Sign is called with the -help parameter, this API outputs a help text with all possible parameters and their descriptions.

3.5.1.2 Functionality: Verify

For the local call an API is initialised, to which the following parameters can be passed.

- source (mandatory): Directory with the files to be verified. The files must first be originated or provided by the third-party application.
- LOG_File (mandatory): File name (e.g. Verify.log) for the log file saved in the dest directory.
- dest (optional): Fully-qualified name of the directory in which the verified files should be saved.
- targetFolderType (optional): Specifies the TargetFolderType as values SAME_AS_SOURCE_FOLDER|ONE_SPECIAL|USE_DEFAULT. SAME_AS_SOURCE_FOLDER cannot be set if a dest has been specified. ONE_SPECIAL is then always used.
- webserviceURL (optional): The URL at which the Governikus Integrated Service is to be made available can be specified here. (e.g. http://208.77.188.166:8088/signer?wsdl)
- checkCertificateStateOnline (optional): This parameter (yes/no/useDefault) can be used to specify with the call whether online verification of the files should be performed. If this parameter is not specified the Signer value is used.
- licenceFile (optional): The licence file to be used for masking can be specified here. The path to the file incl. file name or file URL must be passed for this.
- applicationPolicy (optional): This can be used to specify whether the user interface should be closed or minimised after signature creation (HIDE_AFTER_PROCESS) or should remain open (SHOW_BEYOND). Specified in values: SHOW_BEYOND|HIDE_AFTER_PROCESS|USE_DEFAULT
- processPolicy (optional): Can be used to specify whether the user interface should skip all workflow steps for which the required settings have been completely passed as parameters (SKIPP_COMPLETED_STEP) or whether the workflow should always start with the first (FIRST_STEP) or last (LAST_STEP) step. If the process should be started directly, RUN_PROCESS can be specified. Specified in values: SKIPP_COMPLETED_STEP|FIRST_STEP|LAST_STEP|RUN_PROCESS|USE_DEFAULT
- maxGetWebserviceRetryCount (optional): The number of attempts to create a connection to the web service can be specified here. An interval of 500 ms is left between each attempt to create a connection. If this parameter is not specified the number of attempts is 200.
- maxStartSignerRetryCount (optional): The number of attempts to be made to start Signer can be specified here. An interval of 500 ms is left between each attempt to create a connection. If this parameter is not specified the number of attempts is 100.
- language (optional): The language setting for the application can be passed here. Possible values are de and en. The system language is used as the default.
- Help: Outputs all parameters and return codes with a corresponding description on the console.

	Note: If only this parameter is passed to the application, this terminates after the console output with the return code (100) MISSING_PARAMETER.

- version: Outputs the version and build number on the console (see also note to help).

	Attention: The directory containing the files to be verified and the destination directory must be specified in full by the calling application!

Additional conditions:

- All file operations (source and dest directory) must be locally accessible on the target system.

This API can be called using the two types for each API or via the command line. If the call is made via API, the mandatory parameters must be passed to the constructor with the supplied VerifyParams class. Optional parameters can be passed to the VerifyParams class via setter methods.

If the call is made via the command line, all required parameters must be passed with the call. Call:

```
java -cp CLASSPATH de.bos_bremen.ecard.client.integrated.verify.Verify
-source VALUE [-source VALUE, …]
-LOG_File VALUE
[-dest VALUE]
[-checkCertificateStateOnline VALUE]
[-webserviceURL VALUE]
[-licenceFile VALUE]
[-targetFolderType VALUE]
[-applicationPolicy VALUE]
[-processPolicy VALUE]
[-maxGetWebserviceRetryCount VALUE]
[-maxStartSignerRetryCount VALUE]
[-language VALUE]
```

If Verify is called with the -help parameter, this API outputs a help text with all possible parameters and their descriptions.

3.5.1.3 Functionality: Encrypt

For the local call an API is initialised, to which the following parameters can be passed.

- source (mandatory): Directory with the files to be encrypted. The files must first be originated by the third-party application.
- LOG_File (mandatory): File name (e.g. Encrypt.log) for the log file saved in the dest directory.
- dest (optional): Fully-qualified name of the directory in which the encrypted files should be saved.
- targetFolderType (optional): Specifies the TargetFolderType as values SAME_AS_SOURCE_FOLDER|ONE_SPECIAL|USE_DEFAULT. SAME_AS_SOURCE_FOLDER cannot be set if a dest has been specified. ONE_SPECIAL is then always used.
- createZipArchivBeforeEncrypt (optional): This parameter (yes/no/useDefault) can be used to specify whether the files should be compressed together into a ZIP archive before encryption. If this parameter is not specified the Signer value is used.

- webserviceURL (optional): The URL at which the Governikus Integrated Service is to be made available can be specified here. (e.g. http://208.77.188.166:8088/signer?wsdl)
- serialnumber (optional): Here can be specified which card is to be used for encryption. The serial number of the card must also be given. If USE_DEFAULT is passed as the parameter, Signer uses the last saved setting.
- certificate (optional): The fully-qualified file names of the certificates to be used for encryption can be specified here.
- licenceFile (optional): The licence file to be used for masking can be specified here. The path to the file incl. file name or file URL must be passed for this.
- applicationPolicy (optional): This can be used to specify whether the user interface should be closed or minimised after signature creation (HIDE_AFTER_PROCESS) or should remain open (SHOW_BEYOND). Specified in values: SHOW_BEYOND|HIDE_AFTER_PROCESS|USE_DEFAULT
- processPolicy (optional): Can be used to specify whether the user interface should skip all workflow steps for which the required settings have been completely passed as parameters (SKIPP_COMPLETED_STEP) or whether the workflow should always start with the first (FIRST_STEP) or last (LAST_STEP) step. If the process should be started directly, RUN_PROCESS can be specified. Specified in values: SKIPP_COMPLETED_STEP|FIRST_STEP|LAST_STEP|RUN_PROCESS|USE_DEFAULT
- maxGetWebserviceRetryCount (optional): The number of attempts to be made to connect to the web service can be specified here. An interval of 500 ms is left between each attempt to create a connection. If this parameter is not specified the number of attempts is 200.
- maxStartSignerRetryCount (optional): The number of attempts to be made to start Signer can be specified here. An interval of 500 ms is left between each attempt to create a connection. If this parameter is not specified the number of attempts is 100.
- language (optional): The language setting for the application can be passed here. Possible values are de and en. The system language is used as the default.
- Help: Outputs all parameters and return codes with a corresponding description on the console.

Note: If only this parameter is passed to the application, this terminates after the console output with the return code (100) MISSING_PARAMETER.

- version: Outputs the version and build number on the console (see also note to help).

Attention: The directory containing the files to be encrypted and the destination directory must be specified in full by the calling application!

Additional conditions:

- All file operations (source and destination directory) must be locally accessible on the target system.

This API can be called using JAVA API or via the command line. If the call is made via API, the mandatory parameters must be passed to the constructor with the supplied EncryptParams class. Optional parameters can be passed to the EncryptParams class via setter methods.

If the call is made via the command line, all required parameters must be passed with the call. Call:

```
java -cp CLASSPATH de.bos_bremen.ecard.client.integrated.encrypt.Encrypt
-source VALUE [-source VALUE, …]
-LOG_File VALUE
[-dest VALUE]
[-createZipArchivBeforeEncrypt VALUE]
[-serialnumber VALUE] [-serialnumber VALUE, …]
[-certificate VALUE] [-certificate VALUE, …]
[-webserviceURL VALUE]
[-licenceFile VALUE]
[-targetFolderType VALUE]
[-applicationPolicy VALUE]
[-processPolicy VALUE]
[-maxGetWebserviceRetryCount VALUE]
[-maxStartSignerRetryCount VALUE]
[-language VALUE]
```

If Encrypt is called with the -help parameter, this API outputs a help text with all possible parameters and their descriptions.

3.5.1.4 Functionality: Decrypt

For the local call an API is initialised, to which the following parameters can be passed.

- source (mandatory): Directory with the files to be decrypted. The files must first be originated by the third-party application.
- LOG_File (mandatory): File name (e.g. Decrypt.log) for the log file saved in the dest directory.
- dest (optional): Fully-qualified name of the directory in which the decrypted files should be saved.
- targetFolderType (optional): Specifies the TargetFolderType as values SAME_AS_SOURCE_FOLDER|ONE_SPECIAL|USE_DEFAULT.
  SAME_AS_SOURCE_FOLDER cannot be set if a dest has been specified. ONE_SPECIAL is then always used.
- extractZipArchiv (optional): This parameter (yes/no/useDefault) can be used to specify whether a ZIP archive should be unpacked automatically. If this parameter is not specified the Signer value is used.
- webserviceURL (optional): The URL at which the Governikus Integrated Service is to be made available can be specified here. (e.g. http://208.77.188.166:8088/signer?wsdl)
- serialnumber (optional): The software signature key or card to be used for decryption can be specified here. Either the fully-qualified file name of the software signature key (.p12 file) or the serial number of the card to be used must be specified here. If USE_DEFAULT is passed as the parameter, Signer uses the last saved setting.
- licenceFile (optional): The licence file to be used for masking can be specified here. The path to the file incl. file name or file URL must be passed for this.
- applicationPolicy (optional): This can be used to specify whether the user interface should be closed or minimised after signature creation (HIDE_AFTER_PROCESS) or should remain open (SHOW_BEYOND). Specified in values: SHOW_BEYOND|HIDE_AFTER_PROCESS|USE_DEFAULT
- processPolicy (optional): Can be used to specify whether the user interface should skip all workflow steps for which the required settings have been completely passed as parameters (SKIPP_COMPLETED_STEP) or whether the workflow should always start with the first (FIRST_STEP) or last (LAST_STEP) step. If the process should be started directly,

RUN_PROCESS can be specified. Specified in values: SKIPP_COMPLETED_STEP|FIRST_STEP|LAST_STEP|RUN_PROCESS|USE_DEFAULT

- maxGetWebserviceRetryCount (optional): The number of attempts to be made to connect to the web service can be specified here. An interval of 500 ms is left between each attempt to create a connection. If this parameter is not specified the number of attempts is 200.
- maxStartSignerRetryCount (optional): The number of attempts to be made to start Signer can be specified here. An interval of 500 ms is left between each attempt to create a connection. If this parameter is not specified the number of attempts is 100.
- language (optional): The language setting for the application can be passed here. Possible values are de and en. The system language is used as the default.
- Help: Outputs all parameters and return codes with a corresponding description on the console.

	Note: If only this parameter is passed to the application, this terminates after the console output with the return code (100) MISSING_PARAMETER.

- version: Outputs the version and build number on the console (see also note to help).

	Attention: The directory containing the files to be decrypted and the destination directory must be specified in full by the calling application!

Additional conditions:

- All file operations (source and destination directory) must be locally accessible on the target system.

This API can be called using JAVA API or via the command line. If the call is made via API, the mandatory parameters must be passed to the constructor with the supplied DecryptParams class. Optional parameters can be passed to the DecryptParams class via setter methods.

If the call is made via the command line, all required parameters must be passed with the call. Call:

```
java -cp CLASSPATH de.bos_bremen.ecard.client.integrated.decrypt.Decrypt
-source VALUE [-source VALUE, …]
-LOG_File VALUE
[-dest VALUE]
[-extractZipArchiv VALUE]
[-serialnumber VALUE]
[-webserviceURL VALUE]
[-licenceFile VALUE]
[-targetFolderType VALUE]
[-applicationPolicy VALUE]
[-processPolicy VALUE]
[-maxGetWebserviceRetryCount VALUE]
[-maxStartSignerRetryCount VALUE]
[-language VALUE]
```

If Decrypt is called with the -help parameter, this API outputs a help text with all possible parameters and their descriptions.

3.5.1.5 Functionality: CheckCard

For the local call an API is initialised, to which the following parameters can be passed.

- LOG_File: File name (e.g. CheckCard.log) for the log file saved in the current directory.
- webserviceURL: The URL at which the Governikus Integrated Service is to be made available can be specified here. (e.g. http://208.77.188.166:8088/signer?wsdl)
- serialnumber: The serial number of the card to be used can be specified here.
- maxGetWebserviceRetryCount (optional): The number of attempts to be made to connect to the web service can be specified here. An interval of 500 ms is left between each attempt to create a connection. If this parameter is not specified the number of attempts is ‘200’.
- maxStartSignerRetryCount (optional): The number of attempts to be made to start Signer can be specified here. An interval of 500 ms is left between each attempt to create a connection. If this parameter is not specified the number of attempts is ‘100’.
- Help: Outputs all parameters and return codes with a corresponding description on the console.

Note: If only this parameter is passed to the application, this terminates after the console output with the return code (100) MISSING_PARAMETER.

- version: Outputs the version and build number on the console (see also note to help).

This functionality can be called using API or via the command line. If the call is made via API, the LOG_File, webserviceURL, serialnumber parameters must be passed to the constructor with the supplied CheckCardParams class. Optional parameters do not need to be set.

If the call is made via the command line, the required parameters must be passed with the call. Call:

```
java –cp CLASSPATH de.bos_bremen.ecard.client.integrated.general.CheckCard
[-LOG_File VALUE]
[-serialnumber VALUE]
[-webserviceURL VALUE]
[-maxGetWebserviceRetryCount VALUE]
[-maxStartSignerRetryCount VALUE]
```

If CheckCard is called with the -help parameter, this API outputs a help text with all possible parameters and their descriptions.

3.5.1.6 Functionality: ReinitCards

For the local call an API is initialised, to which the following parameters can be passed.

- LOG_File: File name (e.g. ReinitCards.log) for the log file saved in the current directory.
- webserviceURL: The URL at which the Governikus Integrated Service is to be made available can be specified here. (e.g. http://208.77.188.166:8088/signer?wsdl)
- maxGetWebserviceRetryCount (optional): The number of attempts to be made to connect to the web service can be specified here. An interval of 500 ms is left between each attempt to create a connection. If this parameter is not specified the number of attempts is 200.
- maxStartSignerRetryCount (optional): The number of attempts to be made to start Signer can be specified here. An interval of 500 ms is left between each attempt to create a connection. If this parameter is not specified the number of attempts is 100.
- Help: Outputs all parameters and return codes with a corresponding description on the console.

	Note: If only this parameter is passed to the application, this terminates after the console output with the return code (100) MISSING_PARAMETER.

- version: Outputs the version and build number on the console (see also note to help).

This functionality can be called using API or via the command line. If the call is made via API, the LOG_File and webserviceURL parameters must be passed to the constructor with the supplied ReinitCardsParams class. Optional parameters do not need to be set.

If the call is made via the command line, the required parameters must be passed with the call. Call:

```
java -cp CLASSPATH
de.bos_bremen.ecard.client.integrated.general.ReinitCards                [-
LOG_File VALUE]
[-webserviceURL VALUE]
[-maxGetWebserviceRetryCount VALUE]
[-maxStartSignerRetryCount VALUE]
```

If ReinitCards is called with the -help parameter, this API outputs a help text with all possible parameters and their descriptions.

3.5.1.7 Functionality: GetAvailableCardReader

For the local call an API is initialised, to which the following parameters can be passed.

- LOG_File (optional): File name (e.g. GetAvailableCardReader.log) for the log file saved in the current directory.
- webserviceURL (optional): The URL at which the Governikus Integrated Service is to be made available can be specified here. (e.g. http://208.77.188.166:8088/signer?wsdl)
- dest (optional): The directory in which the certificates should be saved can be specified here. (e.g. d:/temp/).
- maxGetWebserviceRetryCount (optional): The number of attempts to be made to connect to the web service can be specified here. An interval of 500 ms is left between each attempt to create a connection. If this parameter is not specified the number of attempts is ‘200’.
- maxStartSignerRetryCount (optional): The number of attempts to be made to start Signer can be specified here. An interval of 500 ms is left between each attempt to create a connection. If this parameter is not specified the number of attempts is ‘100’.
- Help: Outputs all parameters and return codes with a corresponding description on the console.

	Note: If only this parameter is passed to the application, this terminates after the console output with the return code (100) MISSING_PARAMETER.

- version: Outputs the version and build number on the console (see also note to help).

This functionality can be called using API or via the command line. If the call is made via API, the LOG_File and webserviceURL parameters must be passed to the constructor with the supplied GetAvailableCardReaderParams class. Optional parameters do not need to be set.

If the call is made via the command line, the required parameters must be passed with the call. Call:

```
java -cp CLASSPATH
de.bos_bremen.ecard.client.integrated.general.GetAvailableCardReader
```

```
[-LOG_File VALUE]
[-webserviceURL VALUE]
[-dest VALUE]
[-maxGetWebserviceRetryCount VALUE]
[-maxStartSignerRetryCount VALUE]
```

If GetAvailableCardReader is called with the -help parameter, this API outputs a help text with all possible parameters and their descriptions.

3.5.1.8 Functionality: GetVersion

For the local call an API is initialised, to which the following parameters can be passed.

- LOG_File (optional): File name (e.g. GetVersion.log) for the log file saved in the current directory.
- webserviceURL (optional): The URL at which the Governikus Integrated Service is to be made available can be specified here. (e.g. http://208.77.188.166:8088/signer?wsdl)
- maxGetWebserviceRetryCount (optional): The number of attempts to be made to connect to the web service can be specified here. An interval of 500 ms is left between each attempt to create a connection. If this parameter is not specified the number of attempts is ‘200’.
- maxStartSignerRetryCount (optional): The number of attempts to be made to start Signer can be specified here. An interval of 500 ms is left between each attempt to create a connection. If this parameter is not specified the number of attempts is ‘100’.
- Help: Outputs all parameters and return codes with a corresponding description on the console.

Note: If only this parameter is passed to the application, this terminates after the console output with the return code (100) MISSING_PARAMETER.

- version: Outputs the version and build number on the console (see also note to help).

This functionality can be called using API or via the command line. If the call is made via API, the LOG_File and webserviceURL parameters must be passed to the constructor with the supplied GetVersionParams class. Optional parameters do not need to be set.

If the call is made via the command line, the required parameters must be passed with the call. Call:

```
java -cp CLASSPATH
de.bos_bremen.ecard.client.integrated.general.GetAvailableCardReader
[-LOG_File VALUE]
[-webserviceURL VALUE]
[-maxGetWebserviceRetryCount VALUE]
[-maxStartSignerRetryCount VALUE]
```

If GetVersion is called with the -help parameter, this API outputs a help text with all possible parameters and their descriptions.

3.5.2 Return codes, error handling

The table below contains all the return codes in the product. This also includes some return codes that may be irrelevant to specific projects (e.g. if only software certificates are used).

If multiple errors arise, the application returns the return code with the highest ErrorLevel. If there are multiple return codes with the same ErrorLevel, the first code created is returned. This may mean that KEY_NOT_VALID_YET is returned but the file has still been successfully saved as signed.

A detailed table of all return codes can be found in the bos Signer Developer Manual.

3.6 Structure of the log file

The log file comprises the following entries line by line for each processed file and logged error, in CSV format (; is used as the separator):

- return code for this signature process
- file name (of the original file!)
- return code description
- results file name (on the target system) (, is used as the separator)
- date, time, (dd.MM.yyyy;HH:mm:ss)

3.7 Examples

Signer Integration edition (Sign, Verify, Encrypt and Decrypt) can be called via API or via the command line. If the call is made via API, the parameters must be passed to the constructor using the supplied parameter classes or via the available setter methods. If the call is made via the command line, all required parameters must be passed with the call.

Batch files are available for the Signer Integration Edition command line calls to facilitate accessibility.

The SDK ZIP file contains the following batch files in the Samples folder:

- Signer.bat (sample file with all four functions)
- SmallSigner.bat and SmallVerifier.bat
  - `bos_signer_V2.xml`
  - `sign_licence.xml`
  - `verify_licence.xml`

The following step must be taken before starting SmallSigner/SmallVerifier.bat:

- Check that the %GOVERNIKUS_INSTALL_PATH% variable in the batch files is pointing to the correct directory:
- Copy bos_signer_V2.xml into the user home directory (under Linux ~ and under Windows %userprofile%)

All that is then required is to call SmallSigner.bat or SmallVerifier.bat with the two parameters Source Directory/Source File and Destination Directory.

3.7.1 Sign

API call:

This example is intended to demonstrate the application of the Sign class in signing files. As already described, the mandatory parameters must be passed to the SignParams constructor.

```
SignParams(List<String> source, String sigFormat, String LOG_File)
```

All optional parameters can be set using setter methods. Example:

```
signParams.setDest("D:/dest");
signParams.setLicenceXML("C:/licence.xml");
singParams.setSerienNumber("USE_DEFAULT");
```

etc.

See chapter 3.5.1.1 for a detailed description of the parameters. The following example is intended to show the structure of SignParams:

```
SignParams
```

```
params = new SignParams("D:/source", "ENVELOPED_PKCS_7", "Sign.log");
params.setDest("D:/dest");
params.setEnablePdfInline(BooleanTriState.YES);
params.setKeyStore("D:/key/TestUser1000.p12");
params.setWebServiceURL(http://208.77.188.166:8088/signer?wsdl);
params.setSerialNumber("799150928");
params.setLicenceXML("C:/licence.xml");
params.setApplicationPolicy(ApplicationPolicy.HIDE_AFTER_PROCESS);
params.setProcessPolicy(ProcessPolicy.SKIPP_COMPLETED_STEP);
params.setTargetFolderType(TargetFolderType.ONE_SPECIAL);
params.setMaxGetWebserviceRetryCount(10);
params.setMaxStartSignerRetryCount(5);
params.setLanguage("de");
params.setSignatureAlgorithm(SignatureAlgorithm.RSA_SHA_256);
```

Once the signature parameters are available, the application (GUI) for signing files can be started using the following call:

```
Sign.runSigner(params);
```

The runSigner method starts the application and returns a return code after signature. The return codes are described in more detail in chapter 3.5.2.

Command line call:

The command line call is very similar to the API call. The only difference is that all the required parameters must be included in the call:

```
java -cp CLASSPATH de.bos_bremen.ecard.client.integrated.sign.Sign
-source D:/temp/source
-dest D:/temp/dest
-LOG_File Sign.log
-enablePdfInline NO
-sigFormat ENVELOPED_PKCS_7
-sigPolicy OVERWRITE
-webserviceURL http://208.77.188.166:8088/signer?wsdl
-serialnumber 799150928
-licenceFile C:/licence.xml
-maxStartSignerRetryCount 5
-maxGetWebserviceRetryCount 10
-language de
-signatureAlgorithm rsa_sha_256
```

Optionally, the following Signer Integration Edition call can be used to sign files. The Governikus Integrated Signer default class path is used and does not need to be passed separately.

```
java -jar ./lib/GovernikusSigner.jar
-source D:/temp/source
-dest D:/temp/dest
-LOG_File Sign.log
-enablePdfInline NO
-sigFormat ENVELOPED_PKCS_7
-sigPolicy OVERWRITE
-webserviceURL http://208.77.188.166:8088/signer?wsdl
-serialnumber 08154711
```

```
-licenceFile C:/licence.xml
-maxStartSignerRetryCount 5
-maxGetWebserviceRetryCount 10
-language de
```

Both command line calls shown start the signature function in Signer Integration Edition. If the -help parameter is passed to the examples, the selected command line call outputs a help text with all possible parameters and their descriptions.

3.7.2 Verify

API call:

This example is intended to demonstrate the application of the Verify class in the verification of signed files. All the mandatory parameters must also be passed to the VerifyParams constructor for Verify.

```
VerifyParams(List<String> source, String LOG_File);
```

All optional parameters can be set using setter methods. Example:

```
verifyParams.setDest("D:/dest");
verifyParams.setCheckCertificateStateOnline(BooleanTriState.YES);
```

etc.

See chapter 3.5.1.2 for a detailed description of the parameters. The following example is intended to show the structure of VerifyParams:

```
VerifyParams params = new VerifyParams("D:/source", "Verify.log");
params.setDest("D:/dest");
params.setCheckCertificateStateOnline(BooleanTriState.YES);
params.setWebServiceURL(http://208.77.188.166:8088/signer?wsdl);
params.setLicenceXML("C:/licence.xml");
params.setApplicationPolicy(ApplicationPolicy.HIDE_AFTER_PROCESS);
params.setProcessPolicy(ProcessPolicy.SKIPP_COMPLETED_STEP);
params.setTargetFolderType(TargetFolderType.ONE_SPECIAL);
params.setMaxGetWebserviceRetryCount(10);
params.setMaxStartSignerRetryCount(5);
params.setLanguage("de");
```

Once the verification parameters are available, the application (GUI) for verifying files can be started using the following call:

```
Verify.runVerify(params);
```

The runVerify method starts the application and returns a return code after verification. The return codes are described in more detail in chapter 3.5.2.

Command line call:

The command line call is very similar to the API call. The only difference is that all the required parameters must be included in the call:

```
java –cp CLASSPATH de.bos_bremen.ecard.client.integrated.verify.Verify
-source D:/temp/source
-dest D:/temp/dest
-LOG_File Verify.log
-checkCertificateStateOnline yes
-webserviceURL http://208.77.188.166:8088/signer?wsdl
-licenceFile C:/licence.xml
-maxStartSignerRetryCount 5
-maxGetWebserviceRetryCount 10
-language de
```

The command-line call shown starts verification with Signer Integration Edition. If the -help parameter is passed to the example, the command line call outputs a help text with all possible parameters and their descriptions.

3.7.3 Encrypt

API call:

This example is intended to demonstrate the application of the Encrypt class to encrypt files. All the mandatory parameters must also be passed to the EncryptParams constructor for Encrypt.

```
EncryptParams(List<String> source, String LOG_File)
```

All optional parameters can be set using setter methods. Example:

```
encryptParams.setDest("D:/dest");
encryptParams.setCreateZipArchivBeforeEncrypt(BooleanTriState.YES);
```

etc.

See chapter 3.5.1.3 for a detailed description of the parameters. The following example is intended to show the structure of EncryptParams:

```
List<String> serialnumbers = new ArrayList<String>();
serialnumbers.add("08154711");
serialnumbers.add("08154712");

List<String> certificates = new ArrayList<String>();
certificates.add("D:/Test.cer");

EncryptParams params = new EncryptParams("D:/source", "Encrypt.log");
params.setDest("D:/dest");
params.setCreateZipArchivBeforeEncrypt(BooleanTriState.YES);
params.setSerialNumbers(serialnumbers);
params.setCertificateNumbers(certificates);
params.setWebServiceURL(http://208.77.188.166:8088/signer?wsdl);
params.setLicenceXML("C:/licence.xml");
params.setApplicationPolicy(ApplicationPolicy.HIDE_AFTER_PROCESS);
params.setProcessPolicy(ProcessPolicy.SKIPP_COMPLETED_STEP);
params.setTargetFolderType(TargetFolderType.ONE_SPECIAL);
params.setMaxGetWebserviceRetryCount(10);
params.setMaxStartSignerRetryCount(5);
params.setLanguage("de");
```

Once the encryption parameters are available, the application (GUI) for encrypting files can be started using the following call:

```
Encrypt.runEncrypt(params);
```

The runEncrypt method starts the application and returns a return code after encryption. The return codes are described in more detail in chapter 3.5.2.

Command line call:

The command-line call is very similar to the API call. The only difference is that all the required parameters must be included in the call:

```
java -cp CLASSPATH de.bos_bremen.ecard.client.integrated.encrypt.Encrypt
-source D:/temp/source
-dest D:/temp/dest
-LOG_File Encrypt.log
-createZipArchiv yes
-webserviceURL http://208.77.188.166:8088/signer?wsdl
-serialnumbers 08154711
```

```
-certificates D:/Test.cer
-licenceFile C:/licence.xml
-maxStartSignerRetryCount 5
-maxGetWebserviceRetryCount 10
-language de
```

The command-line call shown starts encryption with Signer Integration Edition. If the -help parameter is passed to the example, the command line call outputs a help text with all possible parameters and their descriptions.

3.7.4 Decrypt

API call:

This example is intended to demonstrate the application of the Decrypt class to decrypt encrypted files. All the mandatory parameters must also be passed to the DecryptParams constructor for Decrypt.

```
DecryptParams(String source, String LOG_File)
```

All optional parameters can be set using setter methods. Example:

```
decryptParams.setDest("D:/dest");
decryptParams.setExtractZipArchiv (BooleanTriState.YES);
decryptParams.setSerienNumber("USE_DEFAULT");
```

etc.

See chapter 3.5.1.4 for a detailed description of the parameters. The following example is intended to show the structure of DecryptParams:

```
DecryptParams params = new DecryptParams("D:/source", "Decrypt.log");
params.setDest("D:/dest");
params.setExtractZipArchiv(BooleanTriState.YES);
params.setSerialNumber("08154711");
params.setWebServiceURL(http://208.77.188.166:8088/signer?wsdl);
params.setLicenceXML("C:/licence.xml");
params.setApplicationPolicy(ApplicationPolicy.HIDE_AFTER_PROCESS);
params.setProcessPolicy(ProcessPolicy.SKIPP_COMPLETED_STEP);
params.setTargetFolderType(TargetFolderType.ONE_SPECIAL);
params.setMaxGetWebserviceRetryCount(10);
params.setMaxStartSignerRetryCount(5);
params.setLanguage("de");
```

Once the decryption parameters are available, the application (GUI) for decrypting encrypted files can be started using the following call:

```
Decrypt.runDecrypt(params);
```

The runDecrypt method starts the application and returns a return code after decryption. The return codes are described in more detail in chapter 3.5.2.

Command line call:

The command line call is very similar to the API call. The only difference is that all the required parameters must be included in the call:

```
java -cp CLASSPATH de.bos_bremen.ecard.client.integrated.decrypt.Decrypt
-source D:/temp/source
-dest D:/temp/dest
-LOG_File Decrypt.log
-extractZipfile yes
-webserviceURL http://208.77.188.166:8088/signer?wsdl
-serialnumbers 08154711
```

```
-licenceFile C:/licence.xml
-maxStartSignerRetryCount 5
-maxGetWebserviceRetryCount 10
-language de
```

The command-line call shown starts decryption with Signer Integration Edition. If the -help parameter is passed to the example, the command line call outputs a help text with all possible parameters and their descriptions.

3.7.5 CheckCard

API call:

This example is intended to show application of the CheckCard class. The mandatory parameters must be passed to the CheckCardParams constructor.

```
CheckCardParams(String logFile, String wsdlURL, String serialnumber)
```

The following example is intended to show the structure of CheckCardParams:

```
CheckCardParams params = new CheckCardParams(

"Sign.log", "http://208.77.188.166:8088/signer?wsdl", "08154711");

params.setMaxGetWebserviceRetryCount(10);
params.setMaxStartSignerRetryCount(5);
```

Once the parameters are available, checking can be started with the following call:

```
CheckCard.run(params);
```

The run method checks the cards and returns a return code. The return codes are described in more detail in chapter 3.5.2.

Command line call:

The command line call is very similar to the API call. The only difference is that all the required parameters must be included in the call:

```
java –cp CLASSPATH de.bos_bremen.ecard.client.integrated.general.CheckCard
-LOG_File Sign.log
-webserviceURL http://208.77.188.166:8088/signer?wsdl
-serialnumbers 08154711
-maxStartSignerRetryCount 5
-maxGetWebserviceRetryCount 10
```

If the -help parameter is passed to the example, the command line call outputs a help text with all possible parameters and their descriptions.

3.7.6 ReinitCards

API call:

This example is intended to show application of the ReinitCards class. The mandatory parameters must be passed to the ReinitCardsParams constructor.

```
ReinitCardsParams(String logFile, String wsdlURL)
```

The following example is intended to show the structure of ReinitCardsParams:

```
ReinitCardsParams params = new ReinitCardsParams(

"Sign.log", "http://208.77.188.166:8088/signer?wsdl");

params.setMaxGetWebserviceRetryCount(10);

params.setMaxStartSignerRetryCount(5);
```

Once the parameters are available, re-initialisation of the cards can be started with the following call:

```
ReinitCards.run(params);
```

The run method resets all cards and returns a return code. The return codes are described in more detail in chapter 3.5.2.

Command line call:

The command line call is very similar to the API call. The only difference is that all the required parameters must be included in the call:

```
java -cp CLASSPATH
de.bos_bremen.ecard.client.integrated.general.ReinitCards
-LOG_File Sign.log
-webserviceURL http://208.77.188.166:8088/signer?wsdl
-maxStartSignerRetryCount 5
-maxGetWebserviceRetryCount 10
```

If the -help parameter is passed to the example, the command line call outputs a help text with all possible parameters and their descriptions.

3.7.7 GetAvailableCardReader

API call:

This example is intended to show application of the GetAvailableCardReader class. The mandatory parameters must be passed to the GetAvailableCardReaderParams constructor.

```
GetAvailableCardReaderParams(String logFile, String wsdlURL)
```

The following example is intended to show the structure of

GetAvailableCardReaderParams:

```
GetAvailableCardReaderParams params = new GetAvailableCardReaderParams(

"Sign.log", "http://208.77.188.166:8088/signer?wsdl");

params.setDestination("D:/temp/");

params.setMaxGetWebserviceRetryCount(10);

params.setMaxStartSignerRetryCount(5);
```

Once the parameters are available, the call can be started:

```
GetAvailableCardReader.run(params);
```

The run method queries all card readers and returns a return code. The return codes are described in more detail in chapter 3.5.2. At the same time, a list of card IDs and the certificate information is written to the submitted LOG file. The certificate information contains the certificate ID, the owner's name and the storage location of the certificate, where a directory has been specified as the destination. These can be analysed if required.

Command line call:

The command line call is very similar to the API call. The only difference is that all the required parameters must be included in the call:

```
java -cp CLASSPATH
de.bos_bremen.ecard.client.integrated.general.GetAvailableCardReader
-LOG_File Sign.log
-dest d:/temp/
-webserviceURL http://208.77.188.166:8088/signer?wsdl
```

```
-maxStartSignerRetryCount 5
-maxGetWebserviceRetryCount 10
```

If the -help parameter is passed to the example, the command line call outputs a help text with all possible parameters and their descriptions.

3.7.8 GetVersion

API call:

This example is intended to show application of the GetVersion class. The mandatory parameters must be passed to the GetVersionParams constructor.

```
GetVersionParams(String logFile, String wsdlURL)
```

The following example is intended to show the structure of GetVersionParams:

GetVersionParams params = new GetVersionParams(

```
"Sign.log", "http://208.77.188.166:8088/signer?wsdl");
```

```
params.setMaxGetWebserviceRetryCount(10);
params.setMaxStartSignerRetryCount(5);
```

Once the parameters are available, the call can be started:

```
GetVersion.run(params);
```

The run method queries the version from the Signer instance to be called and returns this.

Command line call:

The command line call is very similar to the API call. The only difference is that all the required parameters must be included in the call:

```
java -cp CLASSPATH de.bos_bremen.ecard.client.integrated.general.GetVersion
-LOG_File Sign.log
-webserviceURL http://208.77.188.166:8088/signer?wsdl
-maxStartSignerRetryCount 5
-maxGetWebserviceRetryCount 10
```

If the -help parameter is passed to the example, the command line call outputs a help text with all possible parameters and their descriptions.

4 Index of figures

5 Index of tables