



# **DocuSign Signature Appliance Signature API Developer's Guide**

**Version 8.0**

Copyright ©2003-2016 DocuSign, Inc. All rights reserved.

For information about DocuSign trademarks, copyrights and patents refer to the [DocuSign Intellectual Property page](https://www.docusign.com/IP) (<https://www.docusign.com/IP>) on the DocuSign website. All other trademarks and registered trademarks are the property of their respective holders.

No part of this document may be reproduced or transmitted in any form or by any means, electronic or mechanical, for any purpose, without the express written permission of DocuSign, Inc. Under the law, reproducing includes translating into another language or format. Every effort has been made to ensure that the information in this manual is accurate. DocuSign, Inc. is not responsible for printing or clerical errors. Information in this document is subject to change without notice.

DocuSign Signature Appliance Signature API Developer's Guide, version 8

If you have any comments or feedback on our documentation, please send them to us at: [Documentation@DocuSign.com](mailto:Documentation@DocuSign.com).

# Table of Contents

- CoSign Signature APIs Developer’s Guide ..... 1**
- Overview ..... 3**
  - Requirements for Data Authentication Systems.....3
  - Introduction to DocuSign Signature Appliance.....3
    - Environments Supported by DocuSign Signature Appliance.....4
    - Applications that Work with DocuSign Signature Appliance .....4
    - DocuSign Signature Appliance Products and Add-ons .....5
    - Signature APIs.....5
  - DocuSign Signature Appliance Guides .....6
  - Intended Audience.....6
- CoSign Signature Local..... 7**
  - Introduction .....7
  - CoSign Signature Local Components and Capabilities.....7
  - Capabilities of Signing and Validating API .....8
    - Supported File Types .....8
    - Managing Fields Inside a Document.....8
    - Managing Graphical Images .....9
    - Managing User Certificates .....9
    - Signing and Validating Signature Fields in a Document .....9
    - Buffer Signature and Validation Operation .....10
    - User Signing Using a Web Application or an Application Server.....10
    - Advanced Digital Signatures .....10
    - Using in Common Criteria Mode.....11
    - PDF Signature Support .....12
    - Office 2007/2010/2013 Signature Support .....13
    - Word XP/2003 Signature Support .....13
    - TIFF Signature Support .....13
    - InfoPath 2007/2010/2013 Signature Support.....14
    - XML Signature Support.....14
  - Capabilities of User Management API (SAPI-UM).....14
  - Typical CoSign Signature Local Scenarios .....15

Signing Forms using a Desktop Application .....	15
Verifying a Signature in a File .....	15
Signing a File using the Web Services Interface .....	16
Signing a File using the Web Services Interface without Passing the Whole File .....	16
Signing Transaction Data.....	16
Verifying Transaction Data.....	16
Signing Long Raw Data.....	17
Verifying Long Raw Data.....	17
Central File Signing by Different Users.....	17
Batch Signing of Files.....	18
A Workflow Application for Signing a Document Multiple Times .....	18
Introduction to the SAPICOM Interface .....	18
Signature Local Signing/Verifying API .....	21
Using SAPICrypt.....	21
SAPI Session .....	21
SAPIInit .....	23
SAPIHandleAcquire.....	25
SAPIConfigurationValueGet .....	26
SAPIConfigurationValueSet .....	28
SAPIContextRelease.....	30
SAPIStructRelease .....	31
SAPIHandleRelease .....	32
SAPIFinalize .....	33
SAPIExtendedLastErrorGet.....	34
SAPILibInfoGet.....	35
SAPILogonEx .....	36
SAPILogon .....	39
SAPILogoff.....	40
SAPIUserActivate.....	43
SAPICredentialChange .....	45
SAPIGetTokenID.....	47
SAPISetTokenID .....	48
SAPICAInfoGetInit .....	49
SAPICAInfoGetCont .....	53

SAPICACertInstall.....	56
SAPITimeGet.....	58
SAPIGetTokenInfo .....	59
CoSign Signature Local Signature Field .....	60
SAPICreateFileHandleByMem.....	62
SAPICreateFileHandleByName.....	63
SAPIGetFileMemData .....	65
SAPIGetFileHandleType .....	67
SAPISignatureFieldEnumInitEx .....	69
SAPISignatureFieldEnumInit .....	73
SAPISignatureFieldEnumCont .....	74
SAPISignatureFieldInfoGet .....	77
SAPISignatureFieldCreateEx.....	79
SAPISignatureFieldCreate .....	82
SAPISignatureFieldLocatorEnumInit .....	83
SAPISignatureFieldLocatorEnumCont.....	87
SAPISignatureFieldSignEx.....	89
SAPISignatureFieldSign .....	92
SAPISignatureFieldCreateSignEx .....	93
SAPISignatureFieldCreateSign.....	96
SAPISignatureFieldVerify .....	98
SAPIFileIsSignedEx .....	100
SAPIFileIsSigned.....	102
SAPISignatureFieldClear.....	103
SAPISignatureFieldRemove .....	104
SAPISignatureFieldDetailsGUIGet .....	106
SAPISigningCeremonyGUI.....	107
CoSign Signature Local Certificates .....	109
SAPICertificatesEnumInit .....	110
SAPICertificatesEnumCont .....	113
SAPICertificateGetFieldByHandle.....	116
SAPICertificateGetFieldByBlob.....	118
SAPIPKCS7BlobGetValue .....	120
SAPICertificateGUISelect .....	122

SAPICertificateSetDefault .....	124
SAPICertificateGetDefault .....	126
Graphical Image Objects .....	128
Selecting and Retrieving Graphical Image Objects .....	128
Managing Graphical Image Objects.....	129
SAPIGraphicSigImageEnumInit.....	130
SAPIGraphicSigImageEnumCont .....	133
SAPIGraphicSigImageGUISelect.....	136
SAPIGraphicSigImageSetDefaultEx .....	138
SAPIGraphicSigImageSetDefault.....	140
SAPIGraphicSigImageGetDefaultEx.....	141
SAPIGraphicSigImageGetDefault .....	143
SAPIGraphicSigImageInfoGet .....	144
SAPIGraphicSigImageInfoCreate.....	146
SAPIGraphicSigLogoInfoCreate .....	147
SAPIGraphicSigInitialsInfoCreate.....	148
SAPIGraphicSigImageInfoDelete.....	149
SAPIGraphicSigImageInfoUpdate.....	150
SAPIGraphicSigLogoInfoUpdate .....	152
SAPIGraphicSigInitialsInfoUpdate .....	154
CoSign Signature Local Buffer Signing.....	156
SAPIBufferSignEx.....	157
SAPIBufferSign .....	159
SAPIBufferSignInit.....	160
SAPIBufferSignCont .....	164
SAPIBufferSignEndEx .....	167
SAPIBufferSignEnd.....	169
SAPISigningContextPKCS7BlobLenGet .....	170
CoSign Signature Local - Buffer Verify.....	171
SAPIBufferVerifySignature.....	172
SAPIBufferVerifySignatureInit .....	174
SAPIBufferVerifySignatureCont .....	176
SAPIBufferVerifySignatureEnd .....	178
CoSign Signature Local - Enrollment.....	180

SAPIKeyPairGenerate .....	181
SAPIImportCertificate .....	183
CoSign Signature Local - Obsolete Functions.....	185
SAPIGraphicSigImageGet .....	185
SAPIGraphicSigImageSet.....	185
Signature Local User Management API.....	187
Using SAPIUM .....	187
General SAPIUM functions .....	187
Functions for Managing a Single User.....	188
Functions for Managing a Single Group.....	188
Functions for Synchronizing the Users Database with the Application's Users Database .....	189
Functions for Enumerating All the Users Defined in DocuSign Signature Appliance .....	189
Functions for Enumerating All the Groups Defined in DocuSign Signature Appliance .....	189
Functions for Setting Configuration .....	190
SAPIUMInit .....	191
SAPIUMHandleAcquire .....	192
SAPIUMLogonEx.....	193
SAPIUMLogon .....	195
SAPIUMGetTokenID .....	196
SAPIUMSetTokenID .....	197
SAPIUMLogonBuiltIn.....	199
SAPIUMLogoff .....	200
SAPIUMHandleRelease.....	201
SAPIUMFinalize.....	202
SAPIUMExtendedLastErrorGet .....	203
SAPIUMLibInfoGet.....	204
SAPIUMUserAddEx1 .....	205
SAPIUMUserAdd .....	207
SAPIUMUserUpdate .....	208
SAPIUMCredentialSet.....	209
SAPIUMUserSetLogonState .....	211
SAPIUMUserDelete.....	212
SAPIUMUsersSyncBegin .....	213
SAPIUMUserSync .....	214

SAPIUMUsersSyncEnd .....	216
SAPIUMUsersSyncStop .....	217
SAPIUMUsersEnumInitEx .....	218
SAPIUMUsersEnumInit .....	220
SAPIUMUsersEnumCont .....	221
SAPIUMUserInfoGetEx .....	223
SAPIUMUserInfoGet.....	226
SAPIUMCounterReset.....	227
SAPIUMUserAssignGroup.....	228
SAPIUMUserGetByLoginName .....	229
SAPIUMUserTechIDGet.....	230
SAPIUMResetPasswordCounter .....	231
SAPIUMGroupGetByUserGUID .....	232
SAPIUMGroupAdd .....	233
SAPIUMGroupUpdate.....	234
SAPIUMGroupSetStatusByTechID.....	235
SAPIUMGroupDelete.....	236
SAPIUMGroupsEnumInit.....	237
SAPIUMGroupsEnumCont .....	239
SAPIUMGroupGetByTechID.....	241
SAPIUMGroupGetByName .....	242
SAPIUMGroupExtDataUpdate.....	243
SAPIUMGroupExtDataGet .....	244
SAPIUMSCPSet .....	245
Signature Local COM Methods for Signing and Verifying .....	247
SAPICryptCOM Functions.....	247
Init.....	247
HandleAcquire .....	247
ConfigurationValueGet.....	247
ConfigurationValueSet.....	247
ContextRelease.....	247
HandleRelease.....	248
Finalize.....	248
ExtendedLastErrorGet .....	248

LibInfoGet.....	248
LogonEx2.....	248
LogonEx.....	248
Logon .....	248
Logoff .....	249
CredentialChange.....	249
UserActivate .....	249
CAInfoGetInit .....	249
CAInfoGetCont.....	249
CACertInstall .....	250
GetTokenID .....	250
GetTokenInfo .....	250
SetTokenID .....	250
TimeGet .....	250
CreateFileHandleByMem .....	250
CreateFileHandleByName .....	251
GetFileMemData.....	251
GetFileHandleType.....	251
SignatureFieldEnumInitEx.....	251
SignatureFieldEnumInit .....	251
SignatureFieldEnumCont.....	252
SignatureFieldLocatorEnumInit.....	252
SignatureFieldLocatorEnumCont .....	252
SignatureFieldInfoGet.....	252
SignatureFieldCreateEx .....	252
SignatureFieldCreate.....	253
SignatureFieldSignEx2 .....	253
SignatureFieldSignEx .....	253
SignatureFieldSign.....	253
SignatureFieldCreateSignEx2.....	253
SignatureFieldCreateSignEx .....	254
SignatureFieldCreateSign .....	255
SignatureFieldVerify.....	256
FileIsSignedEx .....	256

FileIsSigned .....	256
SignatureFieldClear .....	256
SignatureFieldRemove.....	257
SignatureFieldDetailsGUIGet.....	257
CertificatesEnumInit .....	257
CertificatesEnumCont.....	257
CertificateGetFieldByHandle.....	257
CertificateGetFieldByBlob .....	257
PKCS7BlobGetValue.....	258
CertificateGUISelect.....	258
CertificateSetDefault.....	258
CertificateGetDefault .....	258
GraphicSigImageEnumInit .....	258
GraphicSigImageEnumCont .....	258
GraphicSigImageGUISelect.....	259
SigningCeremonyGUI .....	259
GraphicSigImageSetDefault .....	259
GraphicSigImageSetDefaultEx.....	259
GraphicSigImageGetDefault.....	259
GraphicSigImageGetDefaultEx .....	260
GraphicSigImageInfoGet .....	260
GraphicSigImageInfoCreate .....	260
GraphicSigInitialsInfoCreate .....	260
GraphicSigLogoInfoCreate .....	260
GraphicSigImageInfoDelete .....	260
GraphicSigImageInfoUpdate .....	261
GraphicSigInitialsInfoUpdate .....	261
GraphicSigLogoInfoUpdate.....	261
BufferSignEx2 .....	261
BufferSignEx .....	261
BufferSign.....	262
BufferSignInit .....	262
BufferSignCont.....	262
BufferSignEndEx2.....	262

BufferSignEndEx .....	262
BufferSignEnd .....	262
SigningContextPKCS7BlobLenGet.....	263
BufferVerifySignature .....	263
BufferVerifySignatureInit.....	263
BufferVerifySignatureCont.....	263
BufferVerifySignatureEnd.....	263
LoginTicketCheckInit, LoginTicketCheckCont, and LoginTicketCheckEnd .....	263
SAPICOM Samples.....	265
Sample 1 – Buffer Signing Code Using VB.NET.....	265
Sample 2 – Buffer Verification Code Using VB .NET.....	266
Signature Local COM Methods for User Management.....	269
SAPIUMCOM Functions .....	269
Init.....	269
HandleAcquire .....	269
HandleRelease.....	269
Finalize.....	269
ExtendedLastErrorGet .....	269
LibInfoGet.....	269
Logon .....	269
LogonEx.....	270
Logoff .....	270
GetTokenID .....	270
SetTokenID .....	271
UserAddEx1 .....	271
UserAdd .....	271
UserUpdate .....	271
CredentialSet.....	271
UserSetLogonState.....	272
CounterReset.....	272
UserAssignGroup.....	272
UserDelete.....	272
UsersSyncBegin .....	272
UserSync .....	272

UsersSyncEnd .....	273
UsersSyncStop .....	273
UsersEnumInit .....	273
UsersEnumCont .....	273
UserInfoGet.....	273
UserInfoGetEx .....	274
UserGetByLoginName.....	274
UserTechIDGet .....	274
ResetPasswordCounter.....	274
GroupGetByUserGUID .....	274
GroupAdd .....	275
GroupUpdate.....	275
GroupSetStatusByTechID.....	275
GroupDelete .....	275
GroupGetByTechID .....	275
GroupGetByName .....	275
GroupsEnumInit.....	276
GroupsEnumCont.....	276
GroupExtDataUpdate.....	276
GroupExtDataGet.....	276
SCPSet .....	276

**Appendices ..... 277**

Appendix A: CoSign Signature Local – Signing/Verifying Structures.....	279
SAPI_CONTEXT .....	279
SAPI_SF_TIME_FORMAT_STRUCT.....	281
SAPI_SIG_FIELD_SETTINGS .....	283
SAPI_GRAPHIC_IMAGE_STRUCT .....	286
SAPI_GR_IMG_INFO .....	287
SAPI_FILETIME.....	288
SAPI_SIGNED_FIELD_INFO.....	289
SAPI_INFO_STRUCT .....	291
SAPI_CERT_STATUS_STRUCT .....	292
SAPI_CUSTOM_FIELD_ELEMENTS_STRUCT .....	293
SAPI_TOKEN_VERSION .....	294

SAPI_TOKEN_INFO_STRUCT .....	295
Appendix B: CoSign Signature Local – Signing/Verifying TYPEDEFS .....	297
SAPI_SES_HANDLE.....	297
SAPI_GR_IMG_HANDLE .....	297
SAPI_SIG_FIELD_HANDLE.....	297
SAPI_CERT_HANDLE .....	297
SAPI_FILE_HANDLE .....	297
SAPI_BOOL .....	297
SAPI_PROGRESS_CALLBACK .....	297
Appendix C: CoSign Signature Local – Signing/Verifying ENUMS .....	299
SAPI_ENUM_DRAWING_ELEMENT .....	299
SAPI_ENUM_PROGRESS_CALLBACK_OPERATION.....	300
SAPI_ENUM_CONTEXT_TYPE.....	301
SAPI_ENUM_EXTENDED_TIME_FORMAT.....	302
SAPI_ENUM_DEPENDENCY_MODE.....	303
SAPI_ENUM_SIGNATURE_TYPE.....	304
SAPI_ENUM_GRAPHIC_IMAGE_FORMAT .....	305
SAPI_ENUM_CA_INFO_TYPE .....	306
SAPI_ENUM_FILE_TYPE.....	307
SAPI_ENUM_CONF_ID .....	308
SAPI_ENUM_DATA_TYPE .....	320
SAPI_ENUM_IMAGE_SOURCE_TYPE .....	321
SAPI_ENUM_CERT_STATUS .....	322
SAPI_ENUM_CERT_FIELD.....	323
SAPI_ENUM_PKCS7_FIELD .....	325
SAPI_ENUM_STORE_TYPE.....	326
SAPI_ENUM_STRUCT_TYPE.....	327
SAPI_ENUM_GR_IMG_SELECT_MODE.....	328
SAPI_ENUM_SIG_FIELD_SETTINGS_ID .....	329
SAPI_ENUM_FIELD_MODE .....	331
SAPI_ENUM_SIG_EXT_INFO_FORMAT .....	332
SAPI_ENUM_TICKET_CHECK_STATUS .....	333
SAPI_ENUM_ADVANCED_SIG_LEVEL.....	334
SAPI_ENUM_FILE_HANDLE_TYPE .....	335

SAPI_ENUM_SIG_CLEAR_POLICY .....	336
SAPI_ENUM_AUTH_MODE .....	337
SAPI_ENUM_SERVER_KIND .....	338
SAPI_ENUM_DIRECTORY_KIND .....	339
SAPI_ENUM_PAGE_ROTATE .....	340
Appendix D: CoSign Signature Local – Signing/Verifying Constants and Flags.....	341
General Constants .....	341
Flags when Enumerating Signature Fields.....	343
Flags used by the SAPILogonEx Function .....	344
Flags used by the SAPIGetTokenInfo Function .....	345
Flags when Enumerating Certificates .....	346
Flags when Displaying Certificate List.....	347
Flags when Enumerating or Using Graphical Signatures .....	348
Flags when Using the Signing Ceremony API .....	349
Flags when Creating a Signature Field in PDF Files .....	350
Flags when Signing PDF Files.....	352
Flags when Creating a Signature Field in TIFF Files .....	354
Flags when Signing TIFF Files.....	355
Flags when Using a Signature Field in PDF Documents .....	356
Flags when Using a Signature Field in Word Documents .....	357
Flags when Creating a Signature Field in Office 2007/2010/2013 Documents.....	358
Flags when Signing Word Documents.....	360
Flags when Signing XML Documents.....	361
Time-Stamping/OCSP and Miscellaneous Signature-Related Flags .....	363
Flags when Verifying a Signature.....	365
Flags when Clearing or Removing Signature Fields.....	366
Graphical Signature Image Related Flags.....	367
Custom Fields-Related Constants .....	368
Appendix E: CoSign Signature Local – Signing/Verifying Error Messages .....	369
Appendix F: CoSign Signature Local – User Management Structures .....	393
SAPI_UM_INFO_STRUCT .....	393
SAPI_UM_USERS_FILTER_STRUCT .....	394
SAPI_UM_GROUP_RECORD.....	395
Appendix G: CoSign Signature Local – User Management TYPEDEFS.....	397

SAPI_UM_SES_HANDLE .....	397
SAPI_UM_USR_HANDLE .....	397
SAPI_UM_CONTEXT .....	397
SAPI_UM_GUID .....	397
SAPI_TECH_ID .....	397
SAPI_ENUM_USERS_CONTEXT .....	397
Appendix H: CoSign Signature Local – User Management ENUMS .....	398
SAPI_UM_ENUM_USER_TYPE .....	398
SAPI_UM_ENUM_COUNTER_TYPE .....	399
SAPI_UM_ENUM_USER_LOGIN_STATUS .....	400
SAPI_UM_ENUM_USER_ENROLLMENT_STATUS .....	401
SAPI_UM_ENUM_USER_ENROLLMENT_REASON .....	402
SAPI_UM_ENUM_USER_CERT_STATUS_TYPE .....	403
SAPI_UM_ENUM_PENDING_REQUEST_STATUS_TYPE .....	404
SAPI_UM_ENUM_GROUP_STATUS_TYPE .....	405
SAPI_UM_ENUM_GROUP_KEY_SIZE .....	406
SAPI_UM_ENUM_GROUPS_ORDER_TYPE .....	407
SAPI_UM_ENUM_GROUP_PACKAGE .....	408
SAPI_UM_ENUM_GROUP_FLAGS .....	409
SAPI_UM_ENUM_GROUP_DELETE_USER_OP .....	410
SAPI_UM_ENUM_LOGON_FLAG_TYPE .....	411
SAPI_UM_ENUM_GROUP_VALUE_NAME .....	412
SAPI_UM_USERS_FILTER_EXT_TYPE .....	413
Appendix I: CoSign Signature Local – User Management Constants .....	414
Appendix J: CoSign Signature Local – User Management Error Messages .....	416
<b>CoSign Signature SOAP API .....</b>	<b>420</b>
Introduction .....	421
Technical Information .....	422
Signature and Validation API .....	422
Sign Operation .....	425
Sign Request .....	427
Detailed Descriptions of a Sign Request’s Optional Inputs .....	429
Detailed Descriptions of a Sign Request’s Input Documents .....	439
Sign Response .....	442

Detailed Descriptions of a Sign Response’s Optional Outputs .....	444
Detailed Descriptions of a Sign Response’s Signature Object .....	447
Verify Operation.....	448
Verify Request .....	448
Detailed Descriptions of a Verify Request’s Optional Inputs.....	450
Verify Response.....	451
Detailed Descriptions of a Verify Response’s Optional Outputs.....	453
User Management Operations .....	455
Name Spaces Conventions.....	456
User Management API Operations .....	457
Common User Management Parameters.....	457
RequestType Parameter .....	458
ResponseType Parameter .....	458
listTargets Operations .....	458
add Operation.....	459
modify Operation .....	460
delete Operation .....	461
lookup Operation.....	461
setPassword Operation.....	462
search Operation .....	463
iterate Operation.....	464
closeIterator Operation.....	464
addGroup Operation.....	465
modifyGroup Operation.....	465
deleteGroup Operation.....	466
lookupGroup Operation .....	467
searchGroup Operation .....	468
Using Web Services .....	469
Java .....	469
PHP .....	469
Appendix A: CoSign Signature SOAP Structures.....	471
KeyInfo .....	471
X509DataType.....	471
TimeDateFormatType.....	472

GraphicImageType.....	472
SAPISignedFieldInfoType.....	472
SAPISigFieldSettingsType .....	473
VerificationStatusType .....	474
SPML – TargetType.....	475
SPML – PSOIdentifierType.....	475
SPML – CoSignLogonData .....	475
SPML – UserRecord .....	476
SPML – GroupRecord .....	477
SPML – PSOType.....	477
SPML – ResultsIteratorType .....	478
Appendix B: CoSign Signature SOAP Enumerations .....	479
DependencyModeEnum.....	479
SignatureTypeEnum.....	479
GraphicImageTypeEnum.....	479
ConfIDEnum .....	479
ExtendedTimeFormatEnum .....	481
GraphicImageFormatEnum.....	482
UserCertStatusEnum.....	482
EnrollmentStatusEnum .....	482
UserCertStatusEnum.....	482
UserLoginEnum.....	482
GroupStatusEnum .....	482
PendingRequestStatusEnum .....	482
SPML returnDataType.....	482
<b>CoSign Signature API .....</b>	<b>485</b>
Introduction .....	485
Technical Information .....	485
Signature and Validation API .....	486
User Administration API.....	486
General Data Structures for CoSign Signature API version 0.1.....	487
General Data Structures for CoSign Signature API version 1.0.....	487
SAPIWS_ENUM_PASSWORD_TYPE – Enumerated Values .....	488
Signing and Verifying – V0.1 .....	489

Create and Sign .....	489
Sign .....	493
Get Signature Fields.....	496
Sign Buffer.....	499
Verify Buffer.....	501
Get User Certificates.....	502
Get Graphical Signatures .....	503
Create Graphical Signature .....	505
Delete Graphical Signature .....	507
Validate Credentials.....	508
Change Password.....	509
User Activate .....	510
Get Server Info.....	511
External CA enrollment – V0.1.....	513
Generate key pair .....	513
Import Certificate.....	514
User Administration – V1.0 .....	515
UserAdd.....	515
UserDisable or UserEnable.....	516
UserResetPasswordCounter .....	517
<b>Web Agent API .....</b>	<b>519</b>
Introduction .....	519
Overview of the Signing Process.....	520
Overview of the Verification Process.....	521
Setting Basic Integration Settings in the Web App Configuration File.....	522
Interface between your Web Application and Web Agent.....	523
The Networking Infrastructure .....	523
Uploading a File and Initiating Signing .....	524
Request Format.....	524
Response Format .....	531
Redirecting a User to Web Agent .....	532
Redirecting a User Back to your Web Application.....	533
Downloading a Signed File.....	534
Verifying Signatures in a File .....	536

XML and Redirection URL Samples .....	539
Uploading a File and Initiating Signing – Samples .....	539
Sample Protocol Request.....	539
Sample XML Request .....	539
Sample XML Response .....	540
Redirecting a User to Web Agent – Sample .....	540
Redirecting User Back to your Web Application – Sample.....	540
Downloading a Signed File – Samples .....	540
Uploading a File and Verifying Signatures – Samples .....	541
Appendix A: Web Agent API Error Codes .....	543
Convert to PDF Error Codes .....	543
API Request Error Codes .....	543



# CoSign Signature APIs Developer's Guide

---

CoSign offers a variety of APIs to meet your various digital signature integration requirements:

[CoSign Signature Local](#) – These API functions are implemented as C/C++ libraries and as a COM Object. The library is full featured with all the capabilities for signing files, verifying signatures, and managing signers. The library is highly performant because parts of some API functions are implemented locally.

[CoSign Signature SOAP](#) – This is an HTTPS/SOAP network API which provides a complete set of signing and user management functions. The API offer a full feature set for all scenarios, using any language and any platform.

[CoSign Signature](#) – This is a RESTful web network API. CoSign Signature can be used to easily sign documents and verify signatures.

[Web Agent](#) – Web Agent API is a network API. It provides the fastest and easiest way to add digital signatures to your application. The API includes a full User Experience (UX) layer for signing. Your app communicates with the API via HTTPS calls using XML to pass the signing options.



## Overview

---

Organizations around the world are investing increasingly in business automation solutions to streamline their workflows, improve efficiency and cut costs. Many of these enterprises greatly benefit from their automated processes up to the point where paper must be reintroduced only in order to collect ‘wet ink’ signatures.

The digital signature solution helps enterprises, SMBs and government organizations around the world rapidly transform their signature-dependent processes from paper-intensive to paper-free. Millions of users are proof of how easy it is to quickly achieve ROI by significantly reducing process times and cutting paper-related costs.

## Requirements for Data Authentication Systems

A viable data authentication system must meet the following specifications:

- **Security** – The system must ensure that no one other than the data creator can alter the data in any way.
- **Third-party validation** – The system must enable third parties to validate the authenticity of the data.
- **System independence** – Data authentication must be independent of the system that created the data. Users must be able to validate the authenticity of the data using a known standard independent of any specific system.
- **Validation over time** – Users must be able to validate data authenticity at any point in time.

Currently, the only data authentication method that supports all of these requirements is the Public Key Infrastructure (PKI) method of authenticating data known as “digital signatures”.

## Introduction to DocuSign Signature Appliance

The DocuSign Signature Appliance is a PKI-based, fully featured standard digital signature solution which can be deployed either on-premises or hosted. Both options enable you to quickly establish fully digital signature-dependent processes.

Among its many benefits, DocuSign Signature Appliance provides:

An easy way to sign all major file types directly within widely-used applications, such as [Microsoft Word](#), [Excel](#) and [SharePoint Online](#).

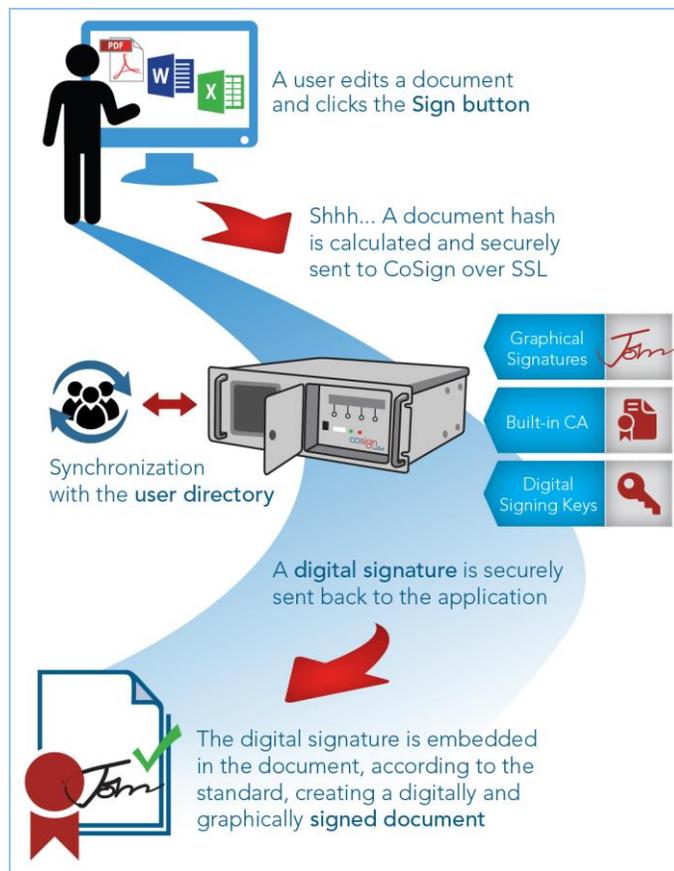
Seamless integration with your existing DM, ECM and workflow systems, including SharePoint, [K2](#), [Nintex](#), [OpenText](#), [Oracle](#), [Alfresco](#) and more.

Signing capabilities anywhere and from any computer, tablet or mobile device.

Reliable verification of the signer’s identity and intent, as well as the document’s integrity.

Enhanced security because sensitive data remains within your IT domain.

[Legally enforceable signatures](#) that are compliant with even the strictest industry-specific regulations and country-specific legislation.



## Environments Supported by DocuSign Signature Appliance

The DocuSign Signature Appliance integrates with leading user management systems, including Microsoft Active Directory and a variety of LDAP (Lightweight Directory Access Protocol) based directories, such as IBM Tivoli. This integration ensures no overhead in managing the digital-signature system and signature credentials (i.e., the private keys that are needed in a PKI environment), solving one of the main problems of legacy digital-signature systems. System managers, network managers, and end-users can continue to use the IT infrastructure in the same manner as before DocuSign Signature Appliance was installed.

DocuSign Signature Appliance stores the signature credentials in a secure server, ensuring that the signer has exclusive access to his or her signature credentials, while still maintaining a centrally managed solution. This is necessary in order to fulfill the security requirement of the data authentication system.

## Applications that Work with DocuSign Signature Appliance

An increasing number of applications can work with DocuSign Signature Appliance as their digital-signature layer without needing any further integration, including:

- Microsoft Office 2007/2010/2013 (Word, Excel, and PowerPoint)
- Microsoft InfoPath 2007/2010/2013
- Adobe Acrobat
- Microsoft SharePoint 2007/2010/2013
- XML

- TIFF files
- Word Perfect
- Microsoft Outlook and Outlook Express
- Adobe Server forms (for signing web forms)
- AutoCAD
- Lotus Notes
- Microsoft BizTalk
- FileNet eForms
- Verity Liquid Office
- ERP systems (e.g., SAP)
- OpenText
- Oracle
- Crystal Reports
- Web applications
- Any application that has a *print* option can use DocuSign Signature Appliance to generate a PDF file and sign it.

For information on using DocuSign Signature Appliance with other applications, contact ARX technical support.

## DocuSign Signature Appliance Products and Add-ons

DocuSign Signature Appliance offers a wide range of products, software tools and add-ons:

- **The DocuSign Signature Appliance** – The appliance includes both hardware and software which are connected to the organization’s network.
- **Client** – The client software is installed on the enterprise’s computers.
- **Administrator** – The administrative software includes the Microsoft Management Console (MMC) snap-in which is installed on the administrative computer.
- **Web App** – This application is deployed in the Microsoft Web Server of the organization and enables users to sign documents in their web browsers without installing any software. Developers can use the Web Agent API to integrate Web App with their applications.
- **DocuSign Signature Appliance Signature APIs** – Developers can use local and network APIs to integrate their applications with appliances.

## Signature APIs

For the 7.1 release, CoSign introduced a new API and renamed its APIs. The new API names are:

New Name	Old Name
CoSign Signature Local	SAPI
CoSign Signature SOAP	SOAP-based Web Services

New Name	Old Name
CoSign Signature	A new API for release 7.1
CoSign Web Agent	CoSign Web App API

“CoSign Signature APIs” refers to the entire set of APIs used with the DocuSign Signature Appliance.

Each API is discussed in its own section of this manual.

## DocuSign Signature Appliance Guides

DocuSign Signature Appliance documentation includes the following guides:

*DocuSign Signature Appliance Administrator Guide* – Provides all the information necessary for an administrator to install and manage the appliance in the various environments in which the DocuSign Signature Appliance can operate.

*DocuSign Signature Appliance Client User Guide* – Provides all the information necessary for an end user to use DocuSign Signature Appliance. Includes information about special add-ins for various applications such as Microsoft Office.

*DocuSign Signature Appliance Signature APIs Developer’s Guide* – Provides all the information necessary for a developer to integrate their application with DocuSign Signature Appliance.

## Intended Audience

This guide is intended for developers looking to integrate desktop applications, enterprise systems, and web applications and services with DocuSign Signature Appliance Digital Signatures capabilities.

# CoSign Signature Local

---

This section includes the following topics:

- [Introduction](#)
- [Signature Local Signing/Verifying API](#)
- [Signature Local User Management API](#)
- [Signature Local COM Methods for Signing and Verifying](#)
- [Signature Local COM Methods for User Management](#)

## Introduction

CoSign Signature Local is an application programming interface (API) that provides digital signature services for desktop applications, enterprise systems, and web applications. In conjunction with DocuSign Signature Appliance, CoSign Signature Local provides a set of functions that allows flexible digital signing of data and documents, avoiding the complexity that is typical for PKI-based digital signatures.

CoSign Signature Local can be used by applications that are not PKI-aware and do not wish to use a full-blown cryptographic API. Developers can use CoSign Signature Local to include support for digital signatures in customized processes, homegrown applications, and integration with third-party applications, systems, and directory services.

This chapter describes the functionality of the CoSign Signature Local API. It is available in two types of programming interfaces:

- *C/C++ libraries for MS Windows applications* – The relevant DLLs are supplied as an add-on to the client.
- *COM Object for Microsoft developers* – This enables COM based developers such as VB or ASP developers to interface with the appliance. Also, this COM object can be used by any .NET development language such as VB.NET or C# to enable digital signature functionality. The COM interface is very similar to the C/C++ libraries interface.

The CoSign Signature Local API includes four components:

- [SAPICrypt](#) provides signing and verification methods.
- [SAPIUM](#) provides user management methods.
- [SAPICryptCOM](#) is the same as SAPICrypt, but provides a COM interface to the methods.
- [SAPIUMCOM](#) is the same as SAPIUM, but provides a COM interface to the methods.

## CoSign Signature Local Components and Capabilities

CoSign Signature Local is comprised of two API components:

- The **signing and validating API**. This API enables signing files and data buffers, managing and retrieving information about the user's certificates, and other signature-related operations. For a detailed overview of this API, refer to [Capabilities of Signing and Validating API](#).
- The **User Management API (SAPI-UM)**. This API enables managing the users and groups, synchronizing users with third-party directory services, and other user management functionality. For an overview of this API, refer to [Capabilities of User Management API \(SAPI-UM\)](#).

## Capabilities of Signing and Validating API

This API enables signing files and data buffers, managing and retrieving information about the user's certificates, and other signature-related operations. This API enables you to:

- Sign/Validate a signature field in a supported application (Word, PDF, etc.).
- Batch sign files (e-Invoices, e-Archiving, etc.).
- Create a signature field in a file (in supported applications).
- Enumerate signature fields in a file (in supported applications).
- Check signature field parameters.
- Enumerate certificates.
- Set the default certificate for a user with multiple certificates.
- Check certificate validity.
- Retrieve certificate fields (such as CN, e-mail, etc.).
- Sign/Validate a single file or data buffer.
- Manage the graphical images of a user that can be embedded inside a graphical signature.

## Supported File Types

This API enables you to sign and verify signature inside the following types of files:

- PDF documents. After performing digital signature, the signatures can be viewed using PDF viewers such as Acrobat Reader.
- Office 2007/2010/2013 documents such as Word 2007/2010/2013 (.docx files) or Excel 2007/2010/2013 (.xlsx files).
- Tiff documents.
- XML files.
- Word XP/2003 documents, where the signature is based on an Entire File signature.
- InfoPath 2007/2010/2013 forms.

The purpose of the signature and validation API is to provide all the necessary tools to perform proper signature and validation operations.

## Managing Fields Inside a Document

As an initial step, it is mandatory to generate empty fields inside a document. These fields, placed in various locations inside the document, will eventually contain the digital signatures. The formats of the field and of the digital signature adhere to the document type. For example, the digital signature field in a PDF file adheres to PDF standards.

**Note:** CoSign Signature Local also supports *Non Visible* digital signatures. These are digital signatures that are not visually embedded inside the document.

The following field management functionality is available:

- Creating and naming a new empty signature field.

- Removing an existing signature field.
- Enumerating all signature fields inside a document.
- Retrieving the information of a signature field.

### **Managing Graphical Images**

DocuSign Signature Appliance can handle multiple graphical images per user. A graphical image is a bitmap image that is based on BMP or JPG format and contains a graphical signature of the end user. In addition, the user account can also include a graphical image for the user's initials or a graphical image for a logo. Starting from CoSign version 5.4, each user is permitted a single logo. Several Graphical signatures/Initials can be kept and incorporated into the signature field during digital signature operation.

The following graphical image management functionality is provided:

- Creating a new graphical image.
- Enumerating existing graphical images of the user.
- Updating an existing graphical image of the user.
- Deleting an existing graphical image.

### **Managing User Certificates**

DocuSign Signature Appliance can handle multiple keys and certificates per user. The X509 certificate contains non-cryptographic parameters such as the Distinguish Name of the user of the certificate, expiration of the certificate, and the name of the CA who provided the certificate.

CoSign Signature Local provides mechanisms for selecting the certificate to be used in digital signature operation, as follows:

- Enumerating all existing certificates.
- Retrieving information from a certificate.
- Specifying the default certificate for the user.

### **Signing and Validating Signature Fields in a Document**

CoSign Signature Local provides the mechanism to sign an empty digital signature field. Upon a successful digital signature operation, the signature field contains the digital signature. The digital signature field is visible in the document.

The following API is provided for signing and validating a digital signature:

- Signature generation API for signing an existing empty field.
- Checking whether a certain field is signed.
- Clearing a field that contains a digital signature.
- Validating an existing signature field.

Starting from CoSign version 5.4, a standard signature ceremony dialog is available in DocuSign Signature Appliance. This dialog is used by all DocuSign Signature Appliance applications and add-ins such as OmniSign and ARX Signature Line Provider for Office 2007/2010/2013. Through the standard signature ceremony dialog, signature components such as reason or graphical signature can be entered or selected.

These signature components are used by the digital signature operation and embedded into the visible signature.

This signature ceremony also supports electronic signatures in PDF files.

### **Buffer Signature and Validation Operation**

CoSign Signature Local provides the mechanism to generate a digital signature upon a given data buffer and verify an existing signature upon a given data buffer.

This mechanism also provide methods for signing a given hash value or validating a signature based on a given hash.

The calling application must specify with which certificate to perform the signature operation.

The generated signature is based on either the PKCS#7 standard or the PKCS#1 standard.

The PKCS#7 or PKCS#1 signature is then validated using the signature validation function.

**Note:** New security regulations recommend using SHA-2 (SHA-256, SHA-384 or SHA-512) as part of the digital signature generation. SHA-2 is supported in CoSign Signature Local for the signature mechanism and the verification mechanism, for machines with XP SP3 or Windows 2003 SP2 and later.

The following API is provided:

- Signature generation API for signing a data buffer.
- Validating an existing signature field.

### **User Signing Using a Web Application or an Application Server**

CoSign Signature Local can be integrated with a web server or application server built on the Windows platform.

To sign data or documents, the end user uses a browser to communicate with the web server. The end user must supply credentials such as a User ID and password. These credentials are used with the SAPILogon API.

Within the Logon/Logoff session, documents are submitted to be signed. The application uses the SAPILogoff function after the signature operations are completed.

In addition, you can execute a third party signature code which is wrapped by the SAPILogon and SAPILogoff API calls. Any existing library that accesses the Microsoft Store and performs a digital signature operation can be activated as part of the user context.

On non-Windows platforms, web applications can use the CoSign Signature and Signature SOAP APIs instead of CoSign Signature Local.

You can use CoSign Signature Local on a PDF file or XML file residing in memory buffers. It is not required to save the file to a web server's disk before using CoSign Signature Local to operate on the saved file.

### **Advanced Digital Signatures**

One of the common usages of digital signatures is to sign documents for long term archiving. This necessitates the following enhancements:

- Including a secure timestamp as part of the digital signature's administrative information.
- Including a certificate status at the time of signing by attaching an OCSP response from the CA as part of the digital signature's administrative information.

The purpose of the secure timestamp is to bind a countable time to the digital signature operation. In this case, you can configure CoSign Signature Local to communicate with a secure timestamp server that provides the secure timestamp. The timestamp is incorporated into the digital signature based on a standard time-stamping protocol called RFC-3161.

As part of the verification procedure, you can use the exact time of the signature to present the signature time, and you can execute a signature validation procedure to evaluate the exact status of the user certificate at the time of the signature.

You can also enhance the created digital signature with current OCSP (Online Certificate Status Protocol) information. OCSP can provide information related to the status of the signing certificate. You can use this indication as part of the signature verification to make sure that the signing certificate was valid at the time when the signature was performed.

**Note:** The embedding of the OCSP information is based on PDF definitions.

For more information on advanced signatures, refer to the following standards:

- For information on CADES (CMS Advanced Electronic Signatures) – refer to RFC 5126.
- For information on XAdES (XML Advanced Electronic Signatures) – refer to <http://www.w3.org/TR/XAdES/>.

### **Using in Common Criteria Mode**

The DocuSign Signature Appliance can be installed in a Common Criteria EAL4+ mode of operation. Common Criteria deployments must be installed in a Directory Independent environment. In this mode of operation, the following additional procedures and activities are required from the end user:

- Any digital signature operation must be authorized by presenting the user's password and an OTP displayed by the user's OTP device.  
This means that any digital signature interface that is used must also be provided with an extended authentication parameter that includes both the static password of the user and a One Time Password (OTP).
- All first-time users must activate their account before they can perform any operation in the account, such as generating a signature key or signing. For that either the client user interface or a User Activation API is required to be used.

### ***User Activation***

In a Common Criteria EAL4+ mode, the first time you wish to use your account you must first perform an activation operation. This operation must be performed only once.

During activation you must supply the given activation password (existing password), the new desired password, and the OTP as it appears in your personal OTP device.

Please make sure you received you activation password and your OTP device from the organization in a secure manner.

**Note:** If you perform an activation procedure and receive a message that the account was already activated, inform the organization immediately.

### ***Protection of Data/Documents during Signing***

It is important to make sure malicious software does not alter the data to be signed between the moment the user initiates a signature operation and until the moment the data is actually signed by the DocuSign

Signature Appliance. This section provides some guidelines for preventing non-trusted software from modifying content to be signed.

- Client installation is signed by *Algorithmic Research's* code signing key. Please make sure that when the product is installed, a relevant message appears indicating that the client software is indeed protected.
- Every DLL and EXE file of the Client is signed with *Algorithmic Research's* code signing key. It is advised to regularly check that the signature is valid by viewing the *digital signatures* tab provided as part of the file properties of the installation DLLs and EXE files.

## PDF Signature Support

The most extensive support for digital signatures is provided for PDF files, mainly due to the rich native support of digital signatures in PDF documents.

All CoSign Signature Local operations conform to the PDF standard (refer to *ISO 32000-1 Document Management – Portable document format – Part 1: PDF 1.7*).

Using CoSign Signature Local, you can create signature fields in a PDF document, sign existing fields, and verify signatures in existing fields. You can also create multiple signature fields, and define whether each signature field is visible.

Starting from CoSign version 6, a PDF file residing in a memory buffer can be used as well as a PDF file saved on the disk.

### PDF Processing Method

Starting from CoSign version 7.4, a new PDF processing mechanism is introduced based on an open source called Podofu (<http://podofu.sourceforge.net/>).

Using the configuration utility, it is possible to switch between the previous PDF processing mechanism and the new PDF processing mechanism (by setting the *PDF Method* parameter in the Signature API > Miscellaneous page).

If the signature process is done through the Web Services interface (either CoSign Signature SOAP or CoSign Signature APIs), the old PDF processing mechanism is used.

### Signature Locators

A new SAPI functionality enables using special textual markups in the PDF document as placeholders for signature fields. The placeholders are called *Signature Locators*. These markups may contain information such as the size of the intended signature field, its time format, etc.

Signature locators are mainly relevant in cases where the initial document is not a PDF file (but a Word file, for example) which the application first converts to a PDF with predefined signature locations and then presents to the user for signing. For example, a predefined Word form that the user fills in, after which it is converted to PDF and presented for signing as a PDF file with predefined signature locations.

Because SAPI does not remove the textual markup from the document, it is recommended to use textual markups that do not appear visibly in the text. For example, use a transparent color for the markup text.

SAPI can handle Signature locators in any of the following ways:

- Using the dedicated enumeration functions [SAPISignatureFieldLocatorEnumInit](#) and [SAPISignatureFieldLocatorEnumCont](#), the application can enumerate the locators and use their information to create new signature fields inside the PDF document
- Using the field enumeration functions [SAPISignatureFieldEnumInitEx](#) and [SAPISignatureFieldEnumCont](#), locators can be listed as regular signature fields and can be normally signed afterwards by the calling application. In this case, the signature operation will

create a new signature field in the location of each Signature Locator and then sign the signature field.

Starting from CoSign version 7.5, OmniSign and the Web App support Signature Locators and present them to the user as regular signature fields.

### **Office 2007/2010/2013 Signature Support**

CoSign Signature Local provides the signature creation and signature verification support for Office 2007/2010/2013 documents based on the Microsoft Signature Line provider standard. The following operations are supported:

- Signing an existing signature field.
- Clearing an existing signature.
- Verifying an existing signature field.

The signing and clearing operations conform to the Office 2007/2010/2013 standard. Thus, you can, for example, sign an Office 2007/2010/2013 document using CoSign Signature Local, and validate the signature using the Office 2007/2010/2013 application.

The certificate that is used for signing must be fully trusted. This means that the certificate chain must be trusted and all CRLs of the certificates in the certificate chain must be accessible.

Failing to trust the entire certificate chain and the certificate CRLs may cause the entire digital signature process to fail.

Starting from CoSign version 6, you can also create an empty signature field and then sign it using any of the following: CoSign Signature Local, the ARX signature line provider for Office 2007/2010/2013, or the Microsoft signature line provider for Office 2007/2010/2013.

Signature fields can be added only in the first page or last page of the document.

### **Word XP/2003 Signature Support**

CoSign Signature Local provides partial support for Word XP/2003 documents. Only the following operations are supported:

- Signing an existing signature field of type *Entire File*.
- Clearing an existing signature.
- Verifying an *existing* signature field.

The *signing and clearing* operations conform to the ARX add-in for Word XP/2003. Thus, you can, for example, sign an Office XP/2003 document using CoSign Signature Local, and validate the signature using the ARX Word add-in.

Since creating an empty signature field is not supported, you will need to prepare in advance empty signature fields inside an Office XP/2003 document and then use CoSign Signature Local to enable applications to sign these signature fields.

### **TIFF Signature Support**

CoSign Signature Local provides extensive support for signing TIFF files, although there is no digital signature standard for TIFF files.

The Tiff tag used by CoSign Signature Local for identifying the digital signature is 50685.

Both visible and invisible signatures are supported. Only one visible signature is supported. The visible signature operation actually replaces the visible content of the TIFF document with the visible signature content.

## InfoPath 2007/2010/2013 Signature Support

CoSign Signature Local provides signature creation and signature verification support for InfoPath 2007/2010 forms. The following operations are supported:

- Signing an existing signature field.
- Clearing an existing signature.
- Verifying an existing signature field.

The signing and clearing operations conform to the InfoPath 2007/2010/2013 standards. This means that you can, for example, sign an InfoPath 2010 document using CoSign Signature Local, and validate the signature using the InfoPath 2010 application.

Because the creation of an empty signature field is not supported by CoSign Signature Local, you will need to prepare in advance empty signature fields in InfoPath 2007/2010/2013 forms and then use CoSign Signature Local to enable applications to sign these signature fields.

The CoSign Signature Local signature conforms with XAdES-BES (refer to <http://www.w3.org/TR/XAdES/>), which is supported in InfoPath 2010/2013 and provides an Advanced Digital Signature format.

## XML Signature Support

Starting from version 5, CoSign Signature Local supports attaching a digital signature to XML content based on the XML digital signature standard, as described in <http://www.w3.org/TR/xmlsig-core/>. In XML, the generated digital signature is not visible. In addition, the digital signature operation generates a signature field and signs it in the same action.

The current implementation enables a single digital signature for a given XML file.

Starting from CoSign version 6, XML data residing in a memory buffer can be used as well as an XML file saved on the disk.

The XML signature standard defines the following three types of digital signatures: Enveloped, Enveloping, and Detached.

The current implementation supports only the Enveloped and Enveloping types.

The basic XML signature is limited and is not useful for long term archival. To address this, a standard called XAdES (XML Advanced Electronic Signature) was defined to enable using XML signatures for long term archival (refer to <http://www.w3.org/TR/XAdES/>). The XML advanced signature standard defines several levels of conformity. CoSign Signature Local supports the following levels: XAdES-BES and XAdES-PES.

## Capabilities of User Management API (SAPI-UM)

This API provides mechanisms for managing users and groups (mainly in a Directory Independent environment), synchronizing users with third-party directory services, and other user management functionality. This API enables:

- Adding, updating and deleting users
- Adding, updating and deleting groups
- Retrieving user information
- Retrieving group information

## Typical CoSign Signature Local Scenarios

The scenarios below are intended to help developers in planning how to activate CoSign Signature Local for their purposes.

**Note:** The term *SAPI session* is relevant only when using the C/C++ based CoSign Signature Local and SAPI-COM.

The idea of a SAPI session is to create a C/C++/COM context in which all CoSign Signature Local operations are executed within the session.

The ability to establish sessions and assign each session to a certain user through a proper login operation enables CoSign Signature Local to provide digital signature API to server applications, where the server application requires support of many users in parallel.

In the case of Web Services deployment, there is no contextual information for the CoSign Signature Local operations.

### Signing Forms using a Desktop Application

This scenario applies to an application which:

- Serves a single user, the user currently logged into the PC.
- Captures the user data into one of a predefined set of forms in which the signature is always located in the same place, regardless of the content entered by the user. Thus, the signature field is actually already created in the form template.

This type of application can use CoSign Signature Local to sign forms, using the following procedure:

1. Create a SAPI session.
2. Find the signature field in the file using the field enumeration function.
3. Sign the signature field.
4. Repeat the above step for all the fields in the document.
5. End the SAPI session.

### Verifying a Signature in a File

This scenario applies to an application that needs to verify a signed file. The application can use CoSign Signature Local as follows:

1. Create a SAPI session (if relevant).
2. Find the signature field in the file using the field enumeration function.
3. Verify the signature field.
4. End the SAPI session (if relevant).

If the file contains more than one signature field, repeat steps 2-3 for each signature field.

You can optionally retrieve the signature field information in case the application wants to display some information about the signature, or in case the application wants to check whether the field is signed before trying to verify its signature. To retrieve signature field information, use the Field information retrieval function.

You can check whether a field is signed without actually verifying its signature or enumerating its signature fields. This can be very helpful for applications wishing to display different icons or colors for different file

types or wishing to offer different menu options according to the file status, without actually performing a verification procedure.

### **Signing a File using the Web Services Interface**

This scenario applies to an application in which the web server communicates with the appliance through the web services interface and provides user credentials as part of the digital signature operation.

This type of application can use CoSign Signature Local to sign files, using the following procedure:

1. Find the signature field in the file using the field enumeration function.
2. Sign the signature field inside the document while providing the signers' credentials. The whole file is transmitted to the DocuSign Signature Appliance.
3. Repeat the above step for all fields in the document.

### **Signing a File using the Web Services Interface without Passing the Whole File**

This scenario applies to an application in which the web server communicates with the DocuSign Signature Appliance through the web services interface and provides user credentials as part of the digital signature operation.

This type of application can use CoSign Signature Local to sign files without passing the whole file, using the following procedure:

1. Find the signature field in the file using the field enumeration function.
2. Compute a hash of the file using any external cryptographic library.
3. Sign the signature field inside the document while providing the signers' credentials. Only the hash of the file is transmitted to the DocuSign Signature Appliance.
4. Repeat the above step for all the fields in the document.

### **Signing Transaction Data**

This scenario applies to an application that needs to sign short raw data, such as transaction data. It can use CoSign Signature Local as follows:

1. Create a SAPI session (if relevant).
2. Sign the raw data using the SAPI Buffer Signing operation.
3. Store the signature data for later verification.
4. End the SAPI session (if relevant).

### **Verifying Transaction Data**

This scenario applies to an application that needs to verify the signature of short raw data, such as transaction data. It can use CoSign Signature Local as follows:

1. Create a SAPI session (if relevant).
2. Verify the signature against the raw data using the SAPI Buffer validation operation.
3. End the SAPI session (if relevant).

## Signing Long Raw Data

This scenario applies to an application that needs to sign long raw data such as long byte streams, binary files, etc.

This type of application can use CoSign Signature Local as follows:

1. Create a SAPI session.
2. Start the signing operation using [SAPIBufferSignInit](#).
3. Sign each portion (i.e., buffer) of data using [SAPIBufferSignCont](#). Call the [SAPIBufferSignCont](#) function as many times as needed, pointing each time to one portion of the data to be signed.
4. End the signing operation and retrieve the signature data using [SAPIBufferSignEndEx](#).
5. Store the signature data for later verification.
6. Sign the raw data using the SAPI Buffer Signing operation.
7. Store the signature data for later verification.
8. End the SAPI session.

## Verifying Long Raw Data

This scenario applies to an application that needs to verify a signature of long raw data such as long byte streams, binary files, etc.

Note: This scenario is relevant only for C/C++ SAPI and SAPI-COM.

This type of application can use CoSign Signature Local as follows:

1. Create a SAPI session. Start the verifying operation using [SAPIBufferVerifySignatureInit](#).
2. Send each portion (i.e., buffer) of data for verification using [SAPIBufferVerifySignatureCont](#). Call the [SAPIBufferVerifySignatureCont](#) function as many times as needed, pointing each time to one portion of the data to be verified.
3. Retrieve the verification result and the signing certificate status using [SAPIBufferVerifySignatureEnd](#).
4. End the SAPI session.

## Central File Signing by Different Users

This scenario applies to an application server or a web application that serves different users, and allows them to sign files of different types using their own certificates and graphical signatures. This type of application can use CoSign Signature Local as follows:

1. Create a SAPI session (if relevant).
2. Logon to DocuSign Signature Appliance with the users credentials using [SAPILogonEx](#) (relevant only to SAPI C/C++ and SAPI-COM).
3. Create a signature field in a specific location in the file. If the file is a PDF or XML file, it can alternatively reside in a memory buffer.
4. Sign the signature field.
5. Repeat steps 3-4 for each additional file you wish to sign.

6. Close the authenticated user session using SAPILogoff (relevant only to SAPI C/C++ and SAPI-COM).
7. End the SAPI session (if relevant).

If the files already contain a signature field, the application can sign the existing field rather than creating a new one.

### Batch Signing of Files

This scenario applies to an application that needs to sign multiple files, either attended or unattended. The application can use CoSign Signature Local as follows:

1. Create a SAPI session (if relevant).
2. For each file that needs to be signed:
  - a. Create a signature field in a specific location in the file.
  - b. Sign the signature field.
3. End the SAPI session (if relevant).

The certificate of the application's owner is used for signing files. For a regular application, the owner is the user under which this application runs. If the application is a service, you can set the user credentials in the service properties, otherwise the service runs with the local system credentials. If the signing certificate belongs to a user other than the application's owner, the application can use **SAPILogon** to logon with these user credentials right after the SAPI session handle is acquired. It then uses **SAPILogoff** to log off when all the files are signed.

In the case of a Web services interface, each file should be sent to be signed in the DocuSign Signature Appliance via the web services interface.

### A Workflow Application for Signing a Document Multiple Times

This scenario applies to an application used for signing a document multiple times by different users having different roles, and which uses a document with existing signature fields. This application can use CoSign Signature Local for each signing operation, as follows:

1. Create a SAPI session (if relevant).
2. Enumerate the signature fields. Check each signature field's information to locate the specific field that needs to be signed according to the workflow logic.
3. Sign the signature field.
4. End the SAPI session (if relevant).

If the workflow logic requires verification of all previously signed fields in the file prior to the signing action, the application can verify the fields based on the API calls described in [Verifying a Signature in a File](#)

### Introduction to the SAPICOM Interface

The SAPICOM interface is very similar to the SAPICrypt interface. Like SAPICrypt, SAPICOM is comprised of the Crypt and User Management API components. The SAPICryptCOM, as well as the SAPIUMCOM, are registered during the installation of the client.

**Note:** The version of SAPICOM and SAPIUMCOM is displayed as 3.21. However, both of these libraries are part of the latest version of the client.

Following are some differences between the SAPICOM and SAPICrypt interfaces:

The `SAPI` prefix of the function name is removed for all of the functions.

All of the unsigned `char*` arrays are replaced by the `SAPIByteArrayInterface` type.

All the `char*` strings and wide `char` strings are replaced with `BSTR*`.

All structures are replaced with an interface with a similar name.

All `void` parameters are replaced with `VARIANT` parameters.

In most cases, the COM data types that are used implicitly indicate the length of the data they store. Therefore, all length parameters are omitted, such as the `ValueLen` parameter in the [SAPIConfigurationValueGet](#) function.

The `progress` callback function cannot be set using the [ConfigurationValueSet](#) method.

The C/C++ [SAPIStructRelease](#) function is not applicable for the COM interface.



## Signature Local Signing/Verifying API

The following sections describe in detail the SAPICrypt Signing and Verifying API.

### Using SAPICrypt

This section provides a detailed explanation of SAPICrypt. SAPICrypt refer to the C/C++ Signature Local Signing/Verifying functions. It explains the basic entities such as a SAPI Session and then explains each SAPICrypt function in detail.

This is followed by typical scenarios and corresponding code samples. These scenarios can help application developers determine how to call the API functions in order to integrate digital signature features in their applications.

The SAPICrypt API .h files are on the SAPI SDK CD, under the `SAPI\include` directory. They include:

- SAPICrypt.h* – Contains all the SAPICrypt API functions.
- SAPIConsts.h* – Contains all the constants used by the SAPICrypt API.
- SAPITypes.h* – Contains all the SAPICrypt API types.
- SAPIEnums.h* – Contains all the SAPICrypt API enumerated types.
- SAPiErrors.h* – Contains all the possible SAPICrypt API error codes.

The SAPICrypt DLL is part of the client installation and is located in the `<Program Files>\ARX\ARX Signature API` directory.

The SAPICrypt library file is part of the SK CD and is located in the `SAPI\lib` directory.

### SAPI Session

The SAPI session functions described in this chapter are general functions intended for various scenarios. These functions can be called by applications that perform signing, verification, signature fields' management, or certificate browsing.

Following are some general usage guidelines:

The [SAPIInit](#) function must *always* be the first function to be called, and [SAPIFinalize](#) must always be called when all work with CoSign Signature Local is completed. After calling [SAPIFinalize](#), the only CoSign Signature Local function that can be called is [SAPIInit](#).

[SAPIHandleAcquire](#) must be called before any other function that uses the SAPI session handle. [SAPIHandleAcquire](#) begins a session that is terminated only by calling [SAPIHandleRelease](#) with the session handle as a parameter. An application can have more than one SAPI session handle opened with CoSign Signature Local, and is usually used by server applications that serve multiple users in parallel. A typical single-threaded CoSign Signature Local application starts by calling [SAPIInit](#) followed by a call to [SAPIHandleAcquire](#). It ends by calling [SAPIHandleRelease](#) followed by [SAPIFinalize](#).

A SAPI session handle is created with default behavior settings. These settings can be retrieved by calling [SAPIConfigurationValueGet](#) with the relevant configuration value ID, and can be modified by calling [SAPIConfigurationValueSet](#). The new configuration values influence all CoSign Signature Local function calls processed after the change. When a session is released, all the modified configuration values are also released. Modifying a SAPI session handle has no impact on other SAPI sessions open now or opened later.

Both Server applications serving multiple different users, as well as single-threaded applications, that wish to avoid the display of dialog boxes call the [SAPILogonEx](#) function. Calling [SAPILogonEx](#) ensures that for the current SAPI session, CoSign Signature Local only uses certificates and graphical images belonging to the user whose credentials were passed to this function. Even in environments in which a user credentials dialog box appears upon calling certain operations (i.e., Directory Independent environments or when the prompt for logon is set), the dialog box does not appear if the user credentials supplied to [SAPILogonEx](#) are correct. Calling [SAPILogoff](#) returns the SAPI session handle to the state it was in before calling [SAPILogonEx](#).

You can invoke third-party libraries that perform digital signature operations within the scope of a user that is defined using the [SAPILogonEx](#) API call. You must set the SAPI\_ENUM\_CONF\_ID\_LOCK\_USER configuration value to 1, so that when the third-party application accesses the Windows Store, it will use certificates that belong to the user that performed the [SAPILogonEx](#) operation.

Some of the CoSign Signature Local functions allocate resources and return them to the application for further use. When these resources are no longer required by the application, they must be released. Resources can be structures, contexts and handles, and the functions that release them are [SAPIStructRelease](#),

SAPIContextRelease and [SAPIHandleRelease](#), respectively.

An application can call [SAPILibInfoGet](#) to find out the CoSign Signature Local version, and thus make use of specific functionality that is implemented only in new versions. The [SAPILibInfoGet](#) function also returns within its information structure an indicator of whether CoSign Signature Local can be called for signing operations, or whether it can be used for signatures verification only.

All the non-void CoSign Signature Local functions return `SAPI_OK` if they succeed and a CoSign Signature Local error code in case of failure. Although the error code usually provides information about which operation failed and why, it is possible and recommended to call [SAPIExtendedLastErrorGet](#) for additional information about the reason for failure.

In a Directory Independent installation, a user can change his user password by calling [SAPICredentialChange](#) with the old and new passwords.

When DocuSign Signature Appliance is deployed in Common Criteria EAL4+ configuration, the user can activate his/her account by calling [SAPIUserActivate](#) with the old and new passwords and an OTP (One Time Password).

Applications wishing to install the CA certificate, or to publish the CRL, call [SAPICAInfoGetInit](#) and [SAPICAInfoGetCont](#) to retrieve the Certificate and CRL data, and then call [SAPICACertInstall](#) to install it in the Windows trust store. Note that in an Active Directory installation, the publication is automatically done by the operating system, and these functions are not called.

In a high availability environment, client applications can use the [SAPIGetTokenID](#) and [SAPISetTokenID](#) to get information about the currently used DocuSign Signature Appliance, or set to which appliance to connect.

You can retrieve the current time from DocuSign Signature Appliance by calling [SAPITimeGet](#).

## SAPIInit

The **SAPIInit** function initializes the SAPI library and must be called before any other function in the library.

```
SAPI int SAPIInit ();
```

### Parameters

This function has no parameters.

### Return Values

If the function succeeds, the return value is `SAPI_OK`.

Some possible error codes are listed in the following table.

Value	Meaning
<code>SAPI_ERR_FAILED_TO_INIT_REG_KEY</code>	Could not find the install path of the SAPI DLL in the registry. A possible reason is CoSign Signature Local is not installed.
<code>SAPI_ERR_FAILED_TO_INIT_LOAD_MODULE</code>	Failed to load the SAPI DLL. A possible reason is that the SAPI DLL was moved from the location specified during CoSign Signature Local installation.

**Remarks**

When no additional CoSign Signature Local function needs to be called, call [SAPIFinalize](#) in order to close the library and free the library resources.

**See Also**

[SAPIFinalize](#)

## SAPIHandleAcquire

The **SAPIHandleAcquire** function acquires a SAPI session handle. This returned handle is then used to make calls to all CoSign Signature Local functions that require such a handle. Configuration values, the default certificate, the graphical image, are all objects that can be set for a specific session handle and influence all subsequent function calls with the handle.

```
SAPI int SAPIHandleAcquire (
    SAPI_SES_HANDLE          *Handle
);
```

### Parameters

#### Handle

[out] Pointer to a handle of a SAPI session.

### Return Values

If the function succeeds, the return value is SAPI\_OK.

Some possible error codes are listed in the following table.

Value	Meaning
SAPI_ERR_NULL_VALUE_IS_NOT_ALLOWED	NULL value for <i>Handle</i> is not allowed.
SAPI_FAILED_TO_ALLOCATE_MEMORY	CoSign Signature Local failed to allocate the necessary memory for the new handle. This might be due to a shortage in system resources.

### Remarks

When the handle is not required anymore, it should be released by calling the [SAPIHandleRelease](#) function. Releasing the session handle frees all the resources related to the handle, and the handle becomes invalid.

### See Also

[SAPIHandleRelease](#), [SAPIConfigurationValueGet](#), [SAPIConfigurationValueSet](#)

## SAPIConfigurationValueGet

The **SAPIConfigurationValueGet** function retrieves one of the configuration values that are defined in [SAPI\\_ENUM\\_CONF\\_ID](#).

```
SAPI int SAPIConfigurationValueGet (
    SAPI_SES_HANDLE          Handle,
    SAPI_ENUM_CONF_ID       ConfValueID,
    SAPI_ENUM_DATA_TYPE     *ConfValueType,
    Void                    *Value,
    unsigned long            *ValueLen,
    SAPI_BOOL                SessionValue
);
```

### Parameters

#### Handle

[in] Handle of the SAPI session created using [SAPIHandleAcquire](#).

#### ConfValueID

[in] The identifier of the value being retrieved.

#### ConfValueType

[out] Pointer to value that receives a code indicating the type of data that is stored as the required configuration value. For a list of the possible type codes, see [SAPI\\_ENUM\\_DATA\\_TYPE](#). The *ConfValueType* parameter can be NULL if the type code is not required.

#### Value

[out] Pointer to the returned configuration value. The *Value* parameter is of the same type as the expected configuration value type. This parameter can be NULL if it is required for querying the length of the returned parameter.

#### ValueLen

[in/out] Pointer to a value that specifies the size, in bytes, of the data type pointed to by the *Value* parameter. When the function returns, this value contains the size of the data copied to *Value*, or, if *Value* is NULL, the size that needs to be allocated for the *Value* parameter.

#### SessionValue

[in] A flag indicating whether the required value is a session value or a global value. Currently only session values are supported, so this parameter must be true.

### Return Values

If the function succeeds, the return value is `SAPI_OK`.

Some possible error codes are listed in the following table.

Value	Meaning
<code>SAPI_ERR_SESSION_IS_NULL</code>	SAPI session handle is NULL, was not created using <a href="#">SAPIHandleAcquire</a> , or is corrupt.
<code>SAPI_ERR_NULL_VALUE_IS_NOT_ALLOWED</code>	<i>ValueLen</i> is NULL and must not be NULL.

Value	Meaning
SAPI_ERR_CONF_VALUE_NOT_FOUND	The value of <i>ConfValueID</i> is not one of the supported configuration ID values.
SAPI_ERR_PARAMETER_LEN_IS_TOO_SHORT	The length of the actual allocated space is shorter than the space needed for proper retrieval of the value.

**Remarks**

The *Value* parameter must be of the same type as the expected configuration value type.

Currently, only session values can be retrieved.

If not set, a default configuration value is returned for each value defined by [SAPI\\_ENUM\\_CONF\\_ID](#).

A configuration value with an ID that is not one of the IDs defined in `SAPI_ENUM_CONF_ID`, can be retrieved only if it was previously set by the [SAPIConfigurationValueSet](#) function.

**See Also**

[SAPIHandleAcquire](#), [SAPIConfigurationValueSet](#)

## SAPIConfigurationValueSet

The **SAPIConfigurationValueSet** function sets one of the configuration values that are defined in [SAPI\\_ENUM\\_CONF\\_ID](#). All subsequent calls with the same SAPI session handle are influenced by the new value set.

```
SAPI int SAPIConfigurationValueSet (
    SAPI_SES_HANDLE           Handle,
    SAPI_ENUM_CONF_ID        ConfValueID,
    SAPI_ENUM_DATA_TYPE      ConfValueType,
    Void                      *Value,
    unsigned long             ValueLen,
    SAPI_BOOL                 SessionValue
);
```

### Parameters

#### Handle

[in] Handle of the SAPI session created using [SAPIHandleAcquire](#).

#### ConfValueID

[in] The identifier of the value being set.

#### ConfValueType

[in] A code indicating the type of data stored as the configuration value. For a list of the possible type codes, see [SAPI\\_ENUM\\_DATA\\_TYPE](#).

#### Value

[in] Pointer to the value of the configuration value being set.

#### ValueLen

[in] A value that specifies the size, in bytes, of the data type pointed to by the *Value* parameter.

#### SessionValue

[in] A flag indicating whether the value is a session value or a global value. Currently only session values are supported, so this parameter must be true.

### Return Values

If the function succeeds, the return value is `SAPI_OK`.

Some possible error codes are listed in the following table.

Value	Meaning
<code>SAPI_ERR_SESSION_IS_NULL</code>	SAPI session handle is NULL, was not created using <a href="#">SAPIHandleAcquire</a> , or is corrupt.
<code>SAPI_ERR_NULL_VALUE_IS_NOT_ALLOWED</code>	<i>Value</i> is NULL and must not be NULL.
<code>SAPI_ERR_FAILED_TO_SET_CONF_VALUE</code>	A general error occurred while setting the configuration value. Call <a href="#">SAPIExtendedLastErrorGet</a> for more information about the exact problem.

**Remarks**

Currently only session values can be set.

If a configuration value is set with an incorrect type, then, when this value is required by other CoSign Signature Local internal functions, the value is ignored and in some cases this can lead to an error.

You can set a configuration value with an ID that is not one of the IDs defined in [SAPI\\_ENUM\\_CONF\\_ID](#), but it cannot be used by any of the CoSign Signature Local internal functions.

**See Also**

[SAPIHandleAcquire](#), [SAPIConfigurationValueGet](#)

## SAPIContextRelease

The **SAPIContextRelease** function releases all the resources related to any kind of CoSign Signature Local context. This function is called whenever the application finishes using a context, to avoid unnecessary allocation of resources.

```
SAPI void SAPIContextRelease (
    SAPI_CONTEXT          *Context
);
```

### *Parameters*

#### *Context*

[in] Any kind of context that was created by any of the CoSign Signature Local context creation functions, such as [SAPICAInfoGetInit](#), [SAPISignatureFieldEnumInitEx](#), [SAPIBufferSignInit](#), [SAPIBufferVerifySignatureInit](#), and [SAPICertificatesEnumInit](#).

### *Return Values*

This function has no return values.

### *Remarks*

Using many contexts without calling the **SAPIContextRelease** function can lead to a shortage of system resources, and in some cases prevent CoSign Signature Local from allocating new resources.

After calling the **SAPIContextRelease** function, you may not use the context anymore. Using a released context can lead to unexpected behavior.

Calling the last function call of an enumeration operation does not automatically release the context. Calling the **SAPIContextRelease** function is mandatory, whether the enumeration operation was successful or not.

### *See Also*

[SAPICAInfoGetInit](#), [SAPISignatureFieldEnumInit](#), [SAPIBufferSignInit](#), [SAPIBufferVerifySignatureInit](#), [SAPICertificatesEnumInit](#)

## SAPIStructRelease

The **SAPIStructRelease** function releases a structure that was previously allocated by a CoSign Signature Local function such as [SAPIGraphicSigImageInfoGet](#), [SAPISignatureFieldInfoGet](#), and [SAPIGraphicSigImageGet](#).

```
SAPI void SAPIStructRelease (  
    void *Struct,  
    SAPI_ENUM_STRUCT_TYPE StructType  
);
```

### Parameters

#### Struct

[in] Pointer to the structure that was allocated by *CoSign Signature Local*

#### StructType

[in] The structure type. The following options are defined:

Parameter value	The structure type pointed by <i>Struct</i>
SAPI_ENUM_STRUCT_TYPE_SIGNED_FIELD	<a href="#">SAPI SIGNED FIELD INFO</a>
SAPI_ENUM_STRUCT_TYPE_GR_IMAGE	<a href="#">SAPI GRAPHIC IMAGE STRUCT</a>
SAPI_ENUM_STRUCT_TYPE_GR_SIG_INFO	<a href="#">SAPI GR IMG INFO</a>
SAPI_ENUM_STRUCT_TYPE_FIELD_INFO	An array of custom fields

### Return Values

This function has no return values.

### Remarks

The structure itself is either statically defined or dynamically allocated by the calling application. Only the structure's fields are allocated and released by *CoSign Signature Local*.

The structures that are allocated by CoSign Signature Local will be the ones released by this function. The application must not try to release the structure fields independently.

### See Also

[SAPIGraphicSigImageInfoGet](#), [SAPISignatureFieldInfoGet](#)

## SAPIHandleRelease

The **SAPIHandleRelease** function releases all the resources related to any kind of SAPI handles. This function is called whenever the application finishes using a handle, to avoid unnecessary allocation of resources.

```
SAPI void SAPIHandleRelease (  
    Void                *Handle  
);
```

### *Parameters*

*Handle*

[in] Any handle that was created by any of the SAPI handle creation functions, such as [SAPIHandleAcquire](#), [SAPISignatureFieldEnumCont](#), [SAPISignatureFieldCreate](#), and [SAPICertificatesEnumCont](#)

### *Return Values*

This function has no return values.

### *Remarks*

Using many handles without calling the **SAPIHandleRelease** function can lead to a shortage of system resources, and in some cases prevent CoSign Signature Local from allocating new resources.

After calling the **SAPIHandleRelease** function, you may not use the handle anymore, and it is recommended to set its value to NULL.

### *See Also*

[SAPIHandleAcquire](#), [SAPISignatureFieldEnumCont](#), [SAPISignatureFieldCreate](#), [SAPICertificatesEnumCont](#)

## **SAPIFinalize**

The **SAPIFinalize** function ends the work with SAPI library, and releases all the global resources of CoSign Signature Local . The **SAPIFinalize** function is called when there are no more calls to CoSign Signature Local functions. Once the **SAPIFinalize** function is called, a call to any of the CoSign Signature Local functions can lead to unexpected results.

```
SAPI int SAPIFinalize ();
```

### ***Parameters***

This function has no parameters.

### ***Return Values***

If the function succeeds, the return value is SAPI\_OK.

### ***See Also***

[SAPIInit](#)

## **SAPIExtendedLastErrorGet**

The **SAPIExtendedLastErrorGet** function retrieves the calling application's extended last-error code value. The extended last-error is often, although not always, different from the error code that was returned in the last function call, and is useful for getting more information about what caused the last function to fail.

```
SAPI int SAPIExtendedLastErrorGet();
```

### ***Parameters***

This function has no parameters.

### ***Return Values***

The return value is the calling application's last error code value. Internal CoSign Signature Local functions set this value whenever an error occurs.

### ***Remarks***

For a complete list of error codes, see the `SAPIErrors.h` file under the `SAPI\include` folder in the SDK CD.

Call the **SAPIExtendedLastErrorGet** function immediately when a function's return value description recommends using this call to return additional data. Immediacy is important because subsequent function calls can reset this value, wiping out the error code set by the most recently failed function.

## SAPILibInfoGet

The **SAPILibInfoGet** function returns some general information about the library.

```
SAPI int SAPILibInfoGet(  
    PSAPI_INFO_STRUCT          SAPIInfoStruct  
);
```

### *Parameters*

*SAPIInfoStruct*

[out] A pointer to the [SAPI\\_INFO\\_STRUCT](#) structure, which holds some general library information.

### *Return Values*

If the function succeeds, the return value is `SAPI_OK`.

## SAPILogonEx

The **SAPILogonEx** function performs a login to DocuSign Signature Appliance using the provided credentials. This function is called in either of the following two scenarios:

In an environment in which DocuSign Signature Appliance automatically displays a login dialog for a new session, but the application wants to avoid this popup, either because it has its own popup or because the operation is not interactive.

When the application runs under certain credentials but wants to login to DocuSign Signature Appliance under different credentials. For example, a web server application that serves many different users.

```
SAPI int SAPILogonEx (
    SAPI_SES_HANDLE          Handle,
    SAPI_LPCWSTR             UserLoginName,
    SAPI_LPCWSTR             DomainName,
    unsigned char            *Password,
    long                     PasswordLen,
    unsigned long            Flags
);
```

### Parameters

#### *Handle*

[in] Handle of the SAPI session created by using [SAPIHandleAcquire](#).

#### *UserLoginName*

[in] The name under which this session is logged in. Note that in a SAML based login, the user name is embedded inside the SAML ticket. Therefore if you are using SAML based authentication, an empty string or the following string *DynamicSlotNoUsername* should be passed as the *UserLoginName* parameter. Any other user ID name (even the user ID inside the SAML token), will fail the **SAPILogonEX** operation.

#### *DomainName*

[in] In an active directory environment – the name of the domain to which UserLoginName belongs.

In a Directory Independent environment or LDAP environment – this parameter is ignored and its value is NULL.

#### *Password*

[in] The password of UserLoginName.

Note that the password value is represented as a wide char string.

#### *PasswordLen*

[in] The length in bytes of the password, including the NULL terminator.

#### *Flags*

[in] Flag values bit mask. For more information, refer to the following section in Appendix D: [Flags used by the SAPILogonEx Function](#).

Note that if you pass the flag SAPI\_LOGON\_FLAG\_TYPE\_DIRECT as part of the *Flags* attribute, all calls to DocuSign Signature Appliance will be executed directly and not through the CAPI/PKCS#11 interfaces, which means that higher performance rates are reached.

Set this option only if you intend to sign buffers, PDF documents, Legacy office documents and Tiff documents.

You can use one of the Auth Mode flags such as `SAPI_LOGON_FLAG_AUTH_MODE_VERIFY_DB_USER_SRV_SIDE` to indicate which authentication mode to use in the specific logon session.

If you pass the flag `SAPI_LOGON_FLAG_AUTH_MODE_SAML_SRV_SIDE` as part of the *Flags* attribute, a SAML based login operation will be carried out. The SAML ticket should be passed as part of the password field. The input password length parameter should be the length in bytes of the given SAML ticket.

If you pass the flag `SAPI_LOGON_FLAG_EXTERNAL_COSIGN_SRV`, then in direct mode – where all calls to DocuSign Signature Appliance are executed directly and not through the CAPI/PKCS#11 interfaces – the client can access an external Server. The IP Address of the designated DocuSign Signature Appliance should be defined by calling the [SAPISetTokenID](#) function.

### **Return Values**

If the function succeeds, the return value is `SAPI_OK`.

Some possible error codes are listed in the following table.

<b>Value</b>	<b>Meaning</b>
<code>SAPI_ERR_SESSION_IS_NULL</code>	SAPI session handle is NULL, was not created using <a href="#">SAPIHandleAcquire</a> , or is corrupt.
<code>SAPI_ERR_TOKEN_HANDLE_NOT_INITIALIZED</code>	SAPI session handle is corrupt.
<code>SAPI_ERR_HANDLE_ALREADY_LOGGED_ON</code>	The session is already logged on. This could be because either: <ul style="list-style-type: none"> <li><input type="checkbox"/> The <code>SAPILogonEx</code> function was already called, and no call to <a href="#">SAPILogoff</a> was made, or</li> <li><input type="checkbox"/> This SAPI session handle was used to access DocuSign Signature Appliance using the default user.</li> </ul>
<code>SAPI_ERR_FAILED_TO_LOGON</code>	An error occurred while trying to logon to DocuSign Signature Appliance. Call <a href="#">SAPIExtendedLastErrorGet</a> for more information about the exact problem.

### **Remarks**

After calling `SAPILogonEx`, it is impossible to use the same SAPI session handle in order to access DocuSign Signature Appliance using the default user. If you do wish to access DocuSign Signature Appliance using the default user, the application must either call [SAPILogoff](#), or create a new SAPI session by calling [SAPIHandleAcquire](#).

If the application performs an operation that accesses DocuSign Signature Appliance with the default user, the application must acquire a new SAPI session before calling the `SAPILogonEx` function, otherwise it causes the `SAPILogonEx` function to fail.

Call [SAPILogoff](#) when there are no other operations that need to be done under the provided user's credentials.

If the `SAPI_LOGON_FLAG_TYPE_DIRECT` flag is used, any of the authentication mode flags such as `SAPI_LOGON_FLAG_AUTH_MODE_VERIFY_DB_USER_SRV_SIDE` cannot be used.

As part of the CoSign appliance version 7.5 performance improvements, if the SAPI\_LOGON\_FLAG\_TYPE\_DIRECT flag is used and you call **SAPILogon** again without calling **SAPILogoff**, SAPI will try reusing the old SSL session for the new logged on user. In this case, the appliance will erase all previous user's session information prior to the new user logon. This functionality is not available when DocuSign Signature Appliance is deployed in Common Criteria mode.

***See Also***

[SAPIHandleAcquire](#), [SAPILogoff](#)

## SAPILogon

The **SAPILogon** function is the older version of the [SAPILogonEx](#) function.

```
SAPI int SAPILogon (  
    SAPI_SES_HANDLE          Handle,  
    SAPI_LPCWSTR            UserLoginName,  
    SAPI_LPCWSTR            DomainName,  
    unsigned char           *Password,  
    long                    PasswordLen,  
);
```

## SAPILogoff

The **SAPILogoff** function ends the session with DocuSign Signature Appliance under a specific user's credentials. The **SAPILogoff** function is only called if the [SAPILogonEx](#) function was previously called, and after ending all work with DocuSign Signature Appliance under the specific user credentials.

```
SAPI int SAPILogoff (
    SAPI_SES_HANDLE          Handle
);
```

### Parameters

*Handle*

[in] Handle of the SAPI session created by using [SAPIHandleAcquire](#) and is the same handle used for the [SAPILogonEx](#) function call.

### Return Values

If the function succeeds, the return value is SAPI\_OK.

### Remarks

It is recommended to call the [SAPIHandleRelease](#) function right after calling the **SAPILogoff** function. After calling the **SAPILogoff** function, any subsequent CoSign Signature Local API call uses DocuSign Signature Appliance with the user default credentials, as if the [SAPILogonEx](#) function was never called.

Calling the **SAPILogoff** function does not automatically release the contexts and handles that were allocated by CoSign Signature Local API functions. These resources must be released by

SAPIContextRelease and [SAPIHandleRelease](#), respectively.

If you skip calling **SAPILogoff** and you interface a version 7.5 CoSign Appliance, the new **SAPILogon** call will use the same SSL session and thus improve the overall system performance. This functionality is not available when the appliance is installed in Common Criteria mode.

***See Also***

[SAPIHandleAcquire](#),

SAPIContextRelease, [SAPIHandleRelease](#), [SAPILogonEx](#)

## SAPIUserActivate

The **SAPIUserActivate** function performs an activation of the user account. This process is mandatory for every account in cases where DocuSign Signature Appliance is deployed in Common Criteria EAL4+ mode.

The function first verifies that the old password (Activation password) is correct and that the OTP is correct, and then stores the credentials calculated from the new password in the DocuSign Signature Appliance users database.

DocuSign Signature Appliance does not store the clear user password but rather a hash value of the password and a unique salt generated by the function, to prevent similar passwords from having similar credential hash values.

```
SAPI int SAPIUserActivate (
    SAPI_SES_HANDLE          Handle,
    SAPI_LPCWSTR             UserLoginName,
    unsigned char            *Password,
    unsigned long            PasswordLen,
    unsigned char            *NewPassword,
    unsigned long            NewPasswordLen,
    unsigned char            *ExtCred,
    unsigned long            ExtCredLen,
    unsigned long            Flags
);
```

### **Parameters**

#### *Handle*

[in] Handle of the SAPI session created by using [SAPIHandleAcquire](#).

#### *UserLoginName*

[in] The name of the user whose credentials are to be changed.

#### *Password*

[in] The user's old password.

Note that the old password value is represented as a wide char string.

#### *PasswordLen*

[in] The length in bytes of the old password, including the NULL terminator.

#### *NewPassword*

[in] The user's new password.

Note that the new password value is represented as a wide char string.

#### *NewPasswordLen*

[in] The length in bytes of the new password, including the NULL terminator.

#### *ExtCred*

[in] The user's OTP (One Time Password).

#### *ExtCredLen*

[in] The length in bytes of the OTP, including the NULL terminator.

## Flags

[in] Reserved for future use.

## Return Values

If the function succeeds, the return value is `SAPI_OK`.

Some possible error codes are listed in the following table.

Value	Meaning
<code>SAPI_ERR_SESSION_IS_NULL</code>	SAPI session handle is NULL, was not created using <a href="#">SAPIHandleAcquire</a> , or is corrupt.
<code>SAPI_ERR_TOKEN_HANDLE_NOT_INITIALIZED</code>	SAPI session handle is corrupt.
<code>SAPI_ERR_NULL_VALUE_IS_NOT_ALLOWED</code>	<i>Password</i> or <i>NewPassword</i> is NULL and must not be NULL.
<code>SAPI_ERR_FAILED_TO_VERIFY_USRPWD</code>	An error occurred while trying to validate the given password. Call <a href="#">SAPIExtendedLastErrorGet</a> for more information about the exact problem.
<code>SAPI_ERR_USER_ALREADY_ACTIVATED</code>	The user was already activated.

## Remarks

This function can be called only in a Directory Independent environment.

## See Also

[SAPIHandleAcquire](#), [SAPIExtendedLastErrorGet](#)

## SAPICredentialChange

The **SAPICredentialChange** function changes a user's credentials. The function first verifies that the old password is correct, and then stores the credentials calculated from the new password in the DocuSign Signature Appliance users database.

DocuSign Signature Appliance does not store the clear user password but rather a hash value of the password and a unique salt generated by the function, to prevent similar passwords from having similar credential hash values.

```
SAPI int SAPICredentialChange (
    SAPI_SES_HANDLE          Handle,
    SAPI_LPCWSTR             UserLoginName,
    unsigned char            *OldPassword,
    unsigned long            OldPasswordLen,
    unsigned char            *NewPassword,
    unsigned long            NewPasswordLen,
    SAPI_CALC_CREDENTIALS_CALLBACK CalcCredentialFunc
);
```

### Parameters

#### *Handle*

[in] Handle of the SAPI session created by using [SAPIHandleAcquire](#).

#### *UserLoginName*

[in] The name of the user whose credentials are to be changed.

#### *OldPassword*

[in] The user's old password.

Note that the old password value is represented as a wide char string.

#### *OldPasswordLen*

[in] The length in bytes of the old password, including the NULL terminator.

#### *NewPassword*

[in] The user's new password.

Note that the new password value is represented as a wide char string.

#### *NewPasswordLen*

[in] The length in bytes of the new password, including the NULL terminator.

#### *CalcCredentialFunc*

[in] A pointer to a user-defined callback function that gets the new password and returns the user credentials. Currently this parameter is not supported, and its value is NULL.

### Return Values

If the function succeeds, the return value is `SAPI_OK`.

Some possible error codes are listed in the following table.

Value	Meaning
SAPI_ERR_SESSION_IS_NULL	SAPI session handle is NULL, was not created using <a href="#">SAPIHandleAcquire</a> , or is corrupt.
SAPI_ERR_TOKEN_HANDLE_NOT_INITIALIZED	SAPI session handle is corrupt.
SAPI_ERR_NULL_VALUE_IS_NOT_ALLOWED	<i>OldPassword</i> or <i>NewPassword</i> is NULL and must not be NULL.
SAPI_ERR_FAILED_TO_UPDATE_CREDENTIAL	An error occurred while trying to change user credentials. Call <a href="#">SAPIExtendedLastErrorGet</a> for more information about the exact problem.

**Remarks**

This function can be called only in a Directory Independent environment.

**See Also**

[SAPIHandleAcquire](#)

## SAPIGetTokenID

The **SAPIGetTokenID** function retrieves a Token ID that identifies the DocuSign Signature Appliance currently used by the SAPI Session. This enables more control over the flow of operations. The function is relevant for cases where several Primary appliances are used and the application would like to control and specify which appliance to use for the digital signature operations.

```
SAPI int SAPIGetTokenID (
    SAPI_SES_HANDLE          Handle,
    char                     *TokenID);
```

### Parameters

#### Handle

[in] Handle of the SAPI session created by using [SAPIHandleAcquire](#).

#### TokenID

[out] The identification of the appliance used by this SAPI Session.

The identification is based on the IP address of the Appliance.

If a DocuSign Signature Appliance Hardware version 8.0+ is used, the IP address may be returned in IPV6 format.

### Return Values

If the function succeeds, the return value is SAPI\_OK.

Some possible error codes are listed in the following table.

Value	Meaning
SAPI_ERR_SESSION_IS_NULL	SAPI session handle is NULL, was not created using <a href="#">SAPIHandleAcquire</a> , or is corrupt.
SAPI_ERR_TOKEN_HANDLE_NOT_INITIALIZED	SAPI session handle is corrupt.

### See Also

[SAPIHandleAcquire](#), [SAPIExtendedLastErrorGet](#), [SAPISetTokenID](#)

## SAPISetTokenID

The **SAPISetTokenID** function sets the Token ID that defines which Appliance should be used by the SAPI Session. This enables more control over the flow of operations. This function is relevant for cases where several Primary appliances are used and the application would like to control and specify which appliance to use for digital signature operations.

The command should be called before the SAPILogon command.

```
SAPI int SAPISetTokenID (
    SAPI_SES_HANDLE          Handle,
    char                      *TokenID);
```

### Parameters

#### Handle

[in] Handle of the SAPI session created by using [SAPIHandleAcquire](#).

#### TokenID

[in] The identification of the appliance that should be used by this SAPI Session.

The identification is based on the IP address of the Appliance.

If a DocuSign Signature Appliance Hardware version 8.0+ is used, the IP address may be specified in IPV6 format.

### Return Values

If the function succeeds, the return value is SAPI\_OK.

Some possible error codes are listed in the following table.

Value	Meaning
SAPI_ERR_SESSION_IS_NULL	SAPI session handle is NULL, was not created using <a href="#">SAPIHandleAcquire</a> , or is corrupt.
SAPI_ERR_TOKEN_HANDLE_NOT_INITIALIZED	SAPI session handle is corrupt.

### See Also

[SAPIHandleAcquire](#), [SAPIExtendedLastErrorGet](#), [SAPIGetTokenID](#)

## SAPICAInfoGetInit

The **SAPICAInfoGetInit** function initializes the continuous operation of retrieving the internal CA publication information. CA publication information can be a CRL or CA certificate. The **SAPICAInfoGetInit** function also returns the CDP (CRL Distribution Point) and AIA (Authority Information Access), according to the information type required.

```
SAPI int SAPICAInfoGetInit (
    SAPI_SES_HANDLE          Handle,
    SAPI_CONTEXT             *CAInfoContext,
    SAPI_LPWSTR              CAPublishLocation,
    long                     *CAPublishLocationLen,
    SAPI_ENUM_CA_INFO_TYPE  CAInfoType
);
```

### Parameters

#### *Handle*

[in] Handle of the SAPI session created by using [SAPIHandleAcquire](#).

#### *CAInfoContext*

[out] Pointer to the SAPICAInfoGet operation context. This context contains internal information about the operation. The information in this context is passed in all subsequent calls to [SAPICAInfoGetCont](#). Changing the value of *CAInfoContext* by the application is not allowed and can lead to unexpected behavior.

*CAInfoContext* must be released by the

SAPIContextRelease function when it is not being used anymore.

### *CAPublishLocation*

[out] A wide char string containing either the CDP or the AIA, as shown in all the certificates created by the CA. The value of *CAPublishLocation* depends on the value of *CAInfoType*. If *CAPublishLocation* is NULL, the function returns the length of the string in *CAPublishLocationLen*.

### *CAPublishLocationLen*

[in/out] Pointer to a value specifying the size, in bytes, of the buffer pointed to by the *CAPublishLocation* parameter. When the function returns, the value contains the number of bytes actually stored or to be stored in the buffer.

### *CAInfoType*

[in] The information type that the SAPICAInfoGet operation returns. The following options are defined:

Parameter value	Content returned by the SAPICAInfoGet operation
SAPI_ENUM_CA_INFO_CRL	The most updated CRL is retrieved. <i>CAPublishLocation</i> contains the CDP – the location where the CRL is published.
SAPI_ENUM_CA_INFO_AIA	The CA certificate is retrieved. <i>CAPublishLocation</i> contains the AIA – the location where the CA certificate is to be published.

### **Return Values**

If the function succeeds, the return value is SAPI\_OK.

Some possible error codes are listed in the following table.

Value	Meaning
SAPI_ERR_SESSION_IS_NULL	SAPI session handle is NULL, was not created using <a href="#">SAPIHandleAcquire</a> , or is corrupt.
SAPI_ERR_NULL_VALUE_IS_NOT_ALLOWED	<i>CAPublishLocationLen</i> or <i>CAInfoContext</i> is NULL and must not be NULL.
SAPI_FAILED_TO_ALLOCATE_MEMORY	CoSign Signature Local failed to allocate the necessary memory for the new operation context. This might be due to a shortage in system resources.
SAPI_ERR_TOKEN_HANDLE_NOT_INITIALIZED	SAPI session handle is corrupt.
SAPI_ERR_FAILED_TO_GET_CA_INFO	A general error occurred while initializing the CAInfo retrieval operation. Call <a href="#">SAPIExtendedLastErrorGet</a> for more information about the exact problem.

### **Remarks**

In an Active Directory environment, DocuSign Signature Appliance automatically publishes its internal CA root certificate as part of the installation process, and the CRL on a daily basis.

In a Directory Independent environment or LDAP environment, it is the application's responsibility to publish the CRL and root certificate in the location returned in *CAPublishLocation*. The value of the publication location is defined during DocuSign Signature Appliance installation.

If the administrator decided to use HTTP as the mechanism for publishing the CA Certificate or CA CRL, it is the application's responsibility to publish the CRL and root certificate in the location returned by *CAPublishLocation*.

***See Also***

[SAPIHandleAcquire](#), [SAPICAInfoGetCont](#),

SAPIContextRelease

## SAPICAInfoGetCont

The **SAPICAInfoGetCont** function continues the CAInfo retrieval operation, and returns a portion of the CA publication information that is to be retrieved. Concatenating all portions of information creates either a valid CRL or a certificate, according to the value of *CAInfoType* in the call to [SAPICAInfoGetInit](#).

```
SAPI int SAPICAInfoGetCont (
    SAPI_SES_HANDLE      Handle,
    SAPI_CONTEXT         *CAInfoContext,
    unsigned char        *CAInfoBlock,
    long                 *CAInfoBlockLen,
    SAPI_BOOL            *LastBlock
);
```

### Parameters

#### *Handle*

[in] Handle of the SAPI session created by using [SAPIHandleAcquire](#).

#### *CAInfoContext*

[in] Pointer to the SAPICAInfoGet operation context created using [SAPICAInfoGetInit](#).

*CAInfoContext* must be released by the

SAPIContextRelease function when it is not being used anymore.

### *CAInfoBlock*

[out] A buffer containing a portion of the information being retrieved. If *CAInfoBlock* is NULL, the function returns the length of the remaining data that still needs to be retrieved.

### *CAInfoBlockLen*

[in/out] Pointer to a value specifying the size, in bytes, of the buffer pointed to by the *CAInfoBlock* parameter. When the function returns, the value contains the number of bytes actually stored, or the remaining data in case *CAInfoBlock* is NULL.

### *LastBlock*

[out] Pointer to a flag indicating that all the data was retrieved, and there is no need for additional calls to the **SAPICAInfoGetCont** function.

### **Return Values**

If the function succeeds, the return value is SAPI\_OK.

Some possible error codes are listed in the following table.

Value	Meaning
SAPI_ERR_SESSION_IS_NULL	SAPI session handle is NULL, was not created using <a href="#">SAPIHandleAcquire</a> , or is corrupt.
SAPI_ERR_NULL_VALUE_IS_NOT_ALLOWED	<i>CAInfoContxt</i> , <i>CAInfoBlockLen</i> or <i>LastBlock</i> is NULL and must not be NULL.
SAPI_ERR_CONTEXT_NOT_INITIALIZED	<i>CAInfoContxt</i> was not initialized correctly, or is corrupt.
SAPI_ERR_TOKEN_HANDLE_NOT_INITIALIZED	SAPI session handle is corrupt.
SAPI_ERR_FAILED_TO_GET_CA_INFO	A general error occurred while getting the CA information. Call <a href="#">SAPIExtendedLastErrorGet</a> for more information about the exact problem.

### **Remarks**

When retrieving the last CA information block, *CAInfoBlock* must point to a memory that is 16 byte longer than the actual required length, and the value of *CAInfoBlockLen* is to be set accordingly.

### **See Also**

[SAPIHandleAcquire](#), [SAPICAInfoGetInit](#),

SAPIContextRelease

## SAPICACertInstall

The **SAPICACertInstall** function installs a certificate in either the Root store or the CA store, within either the user system store or the local machine store, according to the SystemStoreType parameter. If the certificate is a root certificate (i.e., the “issued to” is identical to the “issued by”), it is stored in the Root store. Otherwise, the certificate is an intermediate CA certificate, and is stored in the CA store.

**Note:** To install a certificate in the local machine store, a user must have local administrative privileges and no confirmation dialog pops up. To install a certificate in the user system store, the user needs no special privileges, but a security alert pops up.

```
SAPI int SAPICACertInstall (
    SAPI_SES_HANDLE      Handle,
    unsigned char        *RootCert,
    long                 RootCertLen,
    SAPI_ENUM_STORE_TYPE SystemStoreType
);
```

### Parameters

#### Handle

[in] Handle of the SAPI session created by using [SAPIHandleAcquire](#).

#### RootCert

[in] A buffer containing the data for the certificate to be stored.

#### RootCertLen

[in] The length of the certificate’s data.

#### SystemStoreType

[in] The system store type where the certificate is going to be installed. It can be one of the following values:

Parameter value	The system store where the certificate is to be stored
SAPI_ENUM_STORE_USER	User store. <b>Note:</b> A security alert pops up.
SAPI_ENUM_STORE_LOCAL_MACHINE	Machine store. <b>Note:</b> Requires administrative privileges.

### **Return Values**

If the function succeeds, the return value is `SAPI_OK`.

Some possible error codes are listed in the following table.

<b>Value</b>	<b>Meaning</b>
<code>SAPI_ERR_SESSION_IS_NULL</code>	SAPI session handle is NULL, was not created using <a href="#">SAPIHandleAcquire</a> , or is corrupt.
<code>SAPI_ERR_NULL_VALUE_IS_NOT_ALLOWED</code>	<i>RootCert</i> is NULL and must not be NULL.
<code>SAPI_ERR_ILLEGAL_STORE_TYPE</code>	Only <code>SAPI_ENUM_STORE_USER</code> and <code>SAPI_ENUM_STORE_LOCAL_MACHINE</code> can be used as system store types.
<code>SAPI_ERR_BUFFER_CONTAINS_NO_CERT</code>	The data in <i>RootCert</i> is not valid certificate data.
<code>SAPI_ERR_FAILED_TO_INSTALL_CA_CERT</code>	A general error occurred while installing the CA certificate. Call <a href="#">SAPIExtendedLastErrorGet</a> for more information about the exact problem.

### **See Also**

[SAPIHandleAcquire](#), [SAPICAInfoGetInit](#), [SAPICAInfoGetCont](#)

## SAPITimeGet

The **SAPITimeGet** function retrieves the current time in DocuSign Signature Appliance. If DocuSign Signature Appliance is not available or if the signing certificate is taken from another token, the time retrieved is the time of the local machine.

```
SAPI int SAPITimeGet (
    SAPI_SES_HANDLE          Handle,
    SAPI_FILETIME           *SystemTime
);
```

### Parameters

#### Handle

[in] Handle of the SAPI session created by using [SAPIHandleAcquire](#).

#### SystemTime

[out] The current time in DocuSign Signature Appliance. If DocuSign Signature Appliance is not available or if the signing certificate is taken from another token, *SystemTime* contains the current time of the local machine.

### Return Values

If the function succeeds, the return value is SAPI\_OK.

Some possible error codes are listed in the following table.

Value	Meaning
SAPI_ERR_SESSION_IS_NULL	SAPI session handle is NULL, was not created using <a href="#">SAPIHandleAcquire</a> , or is corrupt.
SAPI_ERR_NULL_VALUE_IS_NOT_ALLOWED	<i>SystemTime</i> is NULL and must not be NULL.
SAPI_ERR_TIME_CONVERT	A general error occurred while retrieving the time. Call <a href="#">SAPIExtendedLastErrorGet</a> for more information about the exact problem.

### See Also

[SAPIHandleAcquire](#)

## SAPIGetTokenInfo

The **SAPIGetTokenInfo** function retrieves information about the DocuSign Signature Appliance used by the SAPI Session.

```
SAPI int SAPIGetTokenInfo (
    SAPI_SES_HANDLE          Handle,
    char                    *TokenID,
    int                     Flags,
    SAPI_TOKEN_INFO_STRUCT *TokenInfo
);
```

### Parameters

#### Handle

[in] Handle of the SAPI session created by using [SAPIHandleAcquire](#).

#### TokenID

[in] The identification of the appliance used by this SAPI Session. For more information refer to [SAPIGetTokenID](#).

#### Flags

[in] Flag values bit mask. For more information, refer to the following section in Appendix D: [Flags used by the SAPIGetTokenInfo Function](#)

#### TokenInfo

[out] Information related to the connected appliance. For more information refer to [SAPI\\_TOKEN\\_INFO\\_STRUCT](#).

### Return Values

If the function succeeds, the return value is SAPI\_OK.

Some possible error codes are listed in the following table.

Value	Meaning
SAPI_ERR_SESSION_IS_NULL	SAPI session handle is NULL, was not created using <a href="#">SAPIHandleAcquire</a> , or is corrupt.

### See Also

[SAPIHandleAcquire](#), [SAPIGetTokenID](#)

## CoSign Signature Local Signature Field

Signing and verifying a file in the context of CoSign Signature Local means signing and verifying a signature field in a file. Thus, a prerequisite to signing a file is to first create signature fields and then sign the fields. When working with signature fields, keep in mind that different file types can behave differently, depending on the different capabilities of the file format and the applications used for viewing the file. For example, you cannot have multiple graphical signatures in a TIFF file.

Typically, there are two alternate methods of signing a signature field:

*If a signature field does not exist:*

Create the signature field using [SAPISignatureFieldCreate](#), and then use the returned signature field handle as input for the [SAPISignatureFieldSign](#) function.

Alternatively, you can use the [SAPISignatureFieldCreateSign](#) function that performs the two operations.

*If the file contains signature fields:*

Enumerate the signature fields using the [SAPISignatureFieldEnumInit](#) and [SAPISignatureFieldEnumCont](#) functions, and then call [SAPISignatureFieldInfoGet](#) to retrieve the signature field information. The signature field information assists you in selecting the field to be signed. Finally, call the [SAPISignatureFieldSign](#) function.

To verify a file, enumerate the signature fields using the [SAPISignatureFieldEnumInit](#) and [SAPISignatureFieldEnumCont](#) functions, then select the field to be verified using [SAPISignatureFieldInfoGet](#) to retrieve the signature field information. Finally, call [SAPISignatureFieldVerify](#).

The [SAPISignatureFieldInfoGet](#) function can also be used for checking whether a signature field is signed (so it can be verified), or not signed (so it can be signed).

Starting from CoSign version 6, when working with PDF or XML file it is possible to read the file from a memory buffer, perform the CoSign Signature Local operations, and then continue to work with the updated memory buffer. For more information see

SAPICreateFileHandleByMem, [SAPICreateFileHandleByName](#), [SAPIGetFileMemData](#) and [SAPIGetFileHandleType](#).

CoSign version 6 also introduces a new functionality in PDF documents, enabling the extraction of location information based on user-embedded field locator strings. This new functionality can enable automated generation of PDF signature fields.

The functions [SAPISignatureFieldLocatorEnumInit](#) and [SAPISignatureFieldLocatorEnumCont](#) enable this functionality. These functions do not delete the field locator strings as part of their functionality.

The [SAPISignatureFieldDetailsGUIGet](#) function displays information of a signed field. The information includes the status of the signature and the status of the signing certificate.

Starting from CoSign version 5.4, you can call a standard signature ceremony dialog to select or enter a signature element that will be encapsulated in the digital signature, as well as embedded in the visible signature. The [SAPISigningCeremonyGUI](#) function displays the signature ceremony dialog.

**Note:** When working with Word XP/2003 files, it is recommended to use MS Word to create and remove the signature fields, and use CoSign Signature Local for all other actions such as signing, verifying, clearing, etc. If you do wish to create or remove a signature field in a Word XP/2003 document using CoSign Signature Local, contact ARX technical support.

**Note:** Starting from CoSign version 6, you can create an empty signature field in a .docx or .xlsx file. The new field can be either an ARX signature line provider field or a Microsoft signature line provider field. You can sign the new signature field using either CoSign Signature Local or Microsoft Office. Additional operations such as signing, verifying or clearing a signature field in .docx/.xlsx files are already supported in previous versions of CoSign Signature Local.

**Note:** When working with Word/Excel 2007/2010/2013 (.docx or .xlsx) files, you must have .Net Framework 3 installed on the machine if you wish to perform operations such as digitally signing an Office 2007/2010/2013 file. To create signature fields, you must have .Net Framework 3.5 installed in the machine.

**Note:** When working with XML files, you must have .Net Framework version 2 or above if you wish to perform operations such as digitally signing an XML file.

## SAPICreateFileHandleByMem

The **SAPICreateFileHandleByMem** function creates a new file handle based on a given memory buffer. This file handle identifies the content of the file in memory and is used by various CoSign Signature Local functions (such as [SAPISignatureFieldEnumInit](#)).

```
SAPI int SAPICreateFileHandleByMem (
    SAPI_FILE_HANDLE          *Handle,
    SAPI_ENUM_FILE_TYPE      FileType,
    unsigned long             Flags,
    unsigned char             *FileMemBuffer,
    unsigned long             FileMemBufferLen
);
```

### Parameters

#### Handle

[out] Output handle that identifies the file content.

#### FileType

[in] File type. Refer to [SAPI\\_ENUM\\_FILE\\_TYPE](#).

#### Flags

[in] Reserved for future use.

#### FileMemBuffer

[in] Pointer to the file content in memory.

#### FileMemBufferLen

[in] The size of the content in memory.

### Return Values

If the function succeeds, the return value is `SAPI_OK`.

Some possible error codes are listed in the following table.

Value	Meaning
<code>SAPI_INVALID_HANDLE</code>	The specified handle is invalid.
<code>SAPI_FAILED_TO_ALLOCATE_MEMORY</code>	CoSign Signature Local failed to allocate the necessary memory for the new handle. This might be due to a shortage in system resources.

### Remarks

When the handle is not required anymore, it should be released by calling the [SAPIHandleRelease](#) function. Releasing the file handle frees all the resources related to the handle, and the handle becomes invalid.

This function is applicable only for PDF files and XML files.

### See Also

[SAPIGetFileMemData](#), [SAPIGetFileHandleType](#), [SAPIHandleRelease](#), [SAPICreateFileHandleByName](#)

## SAPICreateFileHandleByName

The **SAPICreateFileHandleByName** function creates a new file handle based on a given file path. This file handle identifies the file used by the various CoSign Signature Local operations such as the [SAPISignatureFieldEnumInitEx](#) function.

```
SAPI int SAPICreateFileHandleByName (  
    SAPI_FILE_HANDLE          *Handle,  
    SAPI_ENUM_FILE_TYPE       FileType,  
    unsigned long              Flags,  
    SAPI_LPCWSTR               FileUNC  
);
```

### Parameters

#### Handle

[out] Output handle that identifies the file content.

#### FileType

[in] File type. Refer to [SAPI ENUM FILE TYPE](#).

#### Flags

[in] For future use.

#### FileUNC

[in] File name.

### Return Values

If the function succeeds, the return value is `SAPI_OK`.

Some possible error codes are listed in the following table.

Value	Meaning
<code>SAPI_INVALID_HANDLE</code>	Given Handle is invalid.
<code>SAPI_FAILED_TO_ALLOCATE_MEMORY</code>	CoSign Signature Local failed to allocate the necessary memory for the new handle. This might be due to a shortage in system resources.

### Remarks

When the handle is not required anymore, it should be released by calling the [SAPIHandleRelease](#) function. Releasing the file handle frees all the resources related to the handle, and the handle becomes invalid.

### See Also

[SAPIGetFileMemData](#), [SAPIGetFileHandleType](#), [SAPIHandleRelease](#),

SAPICreateFileHandleByMem

## SAPIGetFileMemData

The **SAPIGetFileMemData** function retrieves the updated file content from a given memory buffer.

```
SAPI int SAPIGetFileMemData (  
    SAPI_FILE_HANDLE          *Handle,  
    unsigned long             Flags,  
    unsigned char             *FileMemBuffer,  
    unsigned long             *FileMemBufferLen  
);
```

### Parameters

#### Handle

[in] The given file handle.

#### Flags

[in] Reserved for future use.

#### FileMemBuffer

[out] The application memory buffer that gets the current file content. If this parameter is NULL, the *FileMemBufferLen* parameter will contain the length of the file content in memory.

#### FileMemBufferLen

[in/out] The input must be the size of the allocated *FileMemBuffer*. The output contains the size of the actual content.

### Return Values

If the function succeeds, the return value is `SAPI_OK`.

Some possible error codes are listed in the following table.

Value	Meaning
<code>SAPI_INVALID_HANDLE</code>	The specified handle is invalid.
<code>SAPI_ERR_PARAMETER_LEN_IS_TOO_SHORT</code>	The specified buffer is too short.

### See Also

SAPICreateFileHandleByMem, [SAPIGetFileHandleType](#), [SAPIHandleRelease](#)

## SAPIGetFileHandleType

The **SAPIGetFileHandleType** function retrieves the type of the given file handle.

```
SAPI int SAPIGetFileHandleType (
    SAPI_FILE_HANDLE          *Handle,
    SAPI_ENUM_FILE_HANDLE_TYPE *FileHandleType
);
```

### Parameters

#### Handle

[in] The given file handle.

#### FileHandleType

[out] The type of the given file handle. For more information, refer to [SAPI\\_ENUM\\_FILE\\_HANDLE\\_TYPE](#).

### Return Values

If the function succeeds, the return value is `SAPI_OK`.

Some possible error codes are listed in the following table.

Value	Meaning
<code>SAPI_INVALID_HANDLE</code>	The specified handle is invalid.

### See Also

SAPICreateFileHandleByMem, [SAPICreateFileHandleByName](#)

## SAPISignatureFieldEnumInitEx

The **SAPISignatureFieldEnumInitEx** function initializes the continuous operation of retrieving all the signature fields in a specific file. This operation can also include retrieving all [Signature Locators](#) in the file.

```
SAPI int SAPISignatureFieldEnumInitEx (
    SAPI_SES_HANDLE          Handle,
    SAPI_CONTEXT             *SigFieldContext,
    SAPI_ENUM_FILE_TYPE     FileType,
    SAPI_LPCWSTR             FileUNC,
    SAPI_FILE_HANDLE        FileHandle,
    unsigned long           Flags,
    long                    *SignatureFieldsNum
);
```

### **Parameters**

#### *Handle*

[in] Handle of the SAPI session created by using [SAPIHandleAcquire](#).

#### *SigFieldContext*

[out] Pointer to the signature field enumeration operation context. This context contains internal information about the operation. The information in this context is passed in all subsequent calls to [SAPISignatureFieldEnumCont](#). The application may not change the value of *SigFieldContext*; doing so may lead to unexpected behavior.

*SigFieldContext* must be released by the

SAPIContextRelease function when there is no further need to call the [SAPISignatureFieldEnumCont](#) function.

*FileType*

[in] File type. Refer to [SAPI\\_ENUM\\_FILE\\_TYPE](#).

*FileUNC*

[in] File name. It is recommended to use the *FileHandle* parameter instead and provide "" as an input parameter.

*FileHandle*

[in] File handle. The SAPI file handle to be used for the enumeration operation. The file handle can be created using either the [SAPICreateFileHandleByName](#) function or the

SAPICreateFileHandleByMem function. If this value is NULL, the *FileUNC* parameter is used.

### Flags

[in] Flag values bit mask. For more information, refer to the following section in Appendix D: [Flags when Enumerating Signature Fields](#).

### SignatureFieldsNum

[out] Pointer to a value that holds the number of signature fields that currently exist in the file.

### Return Values

If the function succeeds, the return value is SAPI\_OK.

Some possible error codes are listed in the following table.

Value	Meaning
SAPI_ERR_SESSION_IS_NULL	SAPI session handle is NULL, was not created using <a href="#">SAPIHandleAcquire</a> , or is corrupt.
SAPI_INVALID_HANDLE	The specified handle is invalid.
SAPI_ERR_NULL_VALUE_IS_NOT_ALLOWED	<i>SigFieldContext</i> , <i>SignatureFieldsNum</i> , or <i>FileUNC</i> is NULL and must not be NULL
SAPI_FAILED_TO_ALLOCATE_MEMORY	CoSign Signature Local failed to allocate the necessary memory for the new operation context. This might be due to a shortage in system resources.
SAPI_ERR_FAILED_TO_INIT_ENUM_SF	A general error occurred while initializing the Signature Field Enumeration operation. Call <a href="#">SAPIExtendedLastErrorGet</a> for more information about the exact problem.

### Remarks

The following is relevant if CoSign Signature Local is accessing a file that was saved on the disk (rather than a file residing in a memory buffer): After calling the **SAPISignatureFieldEnumInitEx** function and until the last call to [SAPISignatureFieldEnumCont](#), or until the last usage of one of the signature fields' handles, it is recommended not to change the file using an application other than CoSign Signature Local. Doing so may lead to unexpected behavior.

### See Also

[SAPIHandleAcquire](#), [SAPISignatureFieldEnumCont](#),

SAPIContextRelease

## SAPISignatureFieldEnumInit

The **SAPISignatureFieldEnumInit** function is the older version of the [SAPISignatureFieldEnumInitEx](#) function. In the **SAPISignatureFieldEnumInit** function, a File Handle type is not defined.

```
SAPI int SAPISignatureFieldEnumInit (
    SAPI_SES_HANDLE      Handle,
    SAPI_CONTEXT         *SigFieldContext,
    SAPI_ENUM_FILE_TYPE  FileType,
    SAPI_LPCWSTR         FileUNC,
    unsigned long        Flags,
    long                 *SignatureFieldsNum
);
```

## SAPISignatureFieldEnumCont

The **SAPISignatureFieldEnumCont** function continues the Signature Field Enumeration operation and returns a pointer to a single signature field handle. Every call returns a new Signature Field handle until the end of the list is reached.

```
SAPI int SAPISignatureFieldEnumCont (
    SAPI_SES_HANDLE      Handle,
    SAPI_CONTEXT         *SigFieldContext,
    SAPI_SIG_FIELD_HANDLE *SignatureFieldHandle
);
```

### **Parameters**

#### *Handle*

[in] Handle of the SAPI session created by using [SAPIHandleAcquire](#).

#### *SigFieldContext*

[in] Pointer to the Signature Field Enumeration operation context created using [SAPISignatureFieldEnumInit](#).

*SigFieldContext* must be released by the

SAPIContextRelease function when there is no further need to call the [SAPISignatureFieldEnumCont](#) function.

### *SignatureFieldHandle*

[out] Pointer to a handle of a single signature field. This handle can be used when calling other signature field functions, such as [SAPISignatureFieldInfoGet](#), [SAPISignatureFieldClear](#), [SAPISignatureFieldRemove](#), [SAPISignatureFieldSign](#) and [SAPISignatureFieldVerify](#).

*SignatureFieldHandle* is released by calling the [SAPIHandleRelease](#) function, when there is no further use for the handle.

### **Return Values**

If the function succeeds, the return value is SAPI\_OK.

Some possible error codes are listed in the following table.

Value	Meaning
SAPI_ERR_SESSION_IS_NULL	SAPI session handle is NULL, was not created using <a href="#">SAPIHandleAcquire</a> , or is corrupt.
SAPI_ERR_NULL_VALUE_IS_NOT_ALLOWED	<i>SigFieldContext</i> and <i>SignatureFieldHandle</i> are NULL and must not be NULL.
SAPI_FAILED_TO_ALLOCATE_MEMORY	CoSign Signature Local failed to allocate the necessary memory for the new Signature Field Handle. This might be due to a shortage in system resources.
SAPI_ERR_CONTEXT_NOT_INITIALIZED	The Signature Field Enumeration Context is either not initialized or is corrupt.
SAPI_ERR_FAILED_TO_CONT_ENUM_SF	A general error occurred while retrieving the next signature field handle. Call <a href="#">SAPIExtendedLastErrorGet</a> for more information about the exact problem.

### **See Also**

[SAPIHandleAcquire](#), [SAPISignatureFieldEnumInit](#), [SAPISignatureFieldInfoGet](#), [SAPISignatureFieldClear](#), [SAPISignatureFieldRemove](#), [SAPISignatureFieldSign](#), [SAPISignatureFieldVerify](#),

SAPIContextRelease, [SAPIHandleRelease](#)

## SAPISignatureFieldInfoGet

The **SAPISignatureFieldInfoGet** function returns all the information related to a specific signature field. The information is returned in two structures:

*SignatureFieldSettingStruct* holds data related to the non-cryptographic information, such as appearance.

*SignedFieldInfoStruct* holds crypto data for signed fields.

```
SAPI int SAPISignatureFieldInfoGet (
    SAPI_SES_HANDLE           Handle,
    SAPI_SIG_FIELD_HANDLE     SignatureFieldHandle,
    PSAPI_SIG_FIELD_SETTINGS  SignatureFieldSettingStruct,
    PSAPI_SIGNED_FIELD_INFO   SignedFieldInfoStruct
);
```

### Parameters

#### Handle

[in] Handle of the SAPI session created by using [SAPIHandleAcquire](#).

#### SignatureFieldHandle

[in] The handle of the signature field. This handle was created by [SAPISignatureFieldEnumCont](#).

#### SignatureFieldSettingStruct

[out] Pointer to the [SAPI\\_SIG\\_FIELD\\_SETTINGS](#) structure. This structure holds all the non-crypto information of a signature field.

#### SignedFieldInfoStruct

[out] Pointer to the [SAPI\\_SIGNED\\_FIELD\\_INFO](#) structure. This structure holds the relevant information for signed fields. If the signature field is not signed, the only relevant field in the structure is *IsSigned*. *SignedFieldInfoStruct* is allocated (either statically or dynamically) by the application, only the structure's fields are allocated by the [SAPISignatureFieldInfoGet](#) function and are released by calling the [SAPIStructRelease](#) function.

### Return Values

If the function succeeds, the return value is SAPI\_OK.

Some possible error codes are listed in the following table.

Value	Meaning
SAPI_ERR_SESSION_IS_NULL	SAPI session handle is NULL, was not created using <a href="#">SAPIHandleAcquire</a> , or is corrupt.
SAPI_ERR_NULL_VALUE_IS_NOT_ALLOWED	<i>SignatureFieldHandle</i> , and either <i>SignatureFieldSettingStruct</i> or <i>SignedFieldInfoStruct</i> , are NULL and must not be NULL.

Value	Meaning
SAPI_ERR_FAILED_TO_GET_SIGNATURE_FIELD_INFO	A general error occurred while retrieving the Signature Field information. Call <a href="#">SAPIExtendedLastErrorGet</a> for more information about the exact problem.

**Remarks**

The *SignatureFieldSettingStruct* and *SignedFieldInfoStruct* structures are defined by the application and their pointers are passed as parameters to the function.

**See Also**

[SAPIHandleAcquire](#), [SAPISignatureFieldEnumCont](#), [SAPIStructRelease](#)

## SAPISignatureFieldCreateEx

The **SAPISignatureFieldCreateEx** function creates a single signature field in the specified file. The signature field is created with attributes according to the value of *NewSignatureFieldStruct*. The function optionally returns a handle to the newly created signature field.

```
SAPI int SAPISignatureFieldCreateEx (
    SAPI_SES_HANDLE           Handle,
    SAPI_ENUM_FILE_TYPE      FileType,
    SAPI_LPCWSTR              FileUNC,
    SAPI_FILE_HANDLE         FileHandle,
    PSAPI_SIG_FIELD_SETTINGS NewSignatureFieldStruct,
    unsigned long             Flags,
    SAPI_SIG_FIELD_HANDLE    *SignatureFieldHandle
);
```

### Parameters

#### *Handle*

[in] Handle of the SAPI session created by using [SAPIHandleAcquire](#).

#### *FileType*

[in] File type.

#### *FileUNC*

[in] File name. It is recommended to use the *FileHandle* parameter instead and provide "" as an input parameter.

#### *FileHandle*

[in] File handle. The SAPI file handle to be used for the field creation operation. The file handle can be created using either the [SAPICreateFileHandleByName](#) function or the

SAPICreateFileHandleByMem function. If this value is NULL, the *FileUNC* parameter is used.

### *NewSignatureFieldStruct*

[in] Pointer to the [SAPI\\_SIG\\_FIELD\\_SETTINGS](#) structure. This structure holds all the non-crypto information of the newly created signature field. Note that one of the fields in the structure is the AppearanceMask field which is a bit mask indicating which information is displayed when the signature field is signed (graphical image, signer, reason, date). The mask values are described in [SAPI\\_ENUM\\_DRAWING\\_ELEMENT](#).

### *Flags*

[in] Flag values bit mask. For more information, refer to the following sections in Appendix D: [Flags when Creating a Signature Field in Office 2007/2010/2013 Documents](#) and [Flags when Creating a Signature Field in PDF Files](#).

Note that some signature field flags are passed as part of the function call, while others are passed as part of the signature field [SAPI\\_SIG\\_FIELD\\_SETTINGS](#) structure to which the function points.

The relevant sections in [Appendix D](#) note which flags are passed as part of the function and which are passed as part of the signature field structure.

### *SignatureFieldHandle*

[out] Pointer to a handle of the newly created signature field. If *SignatureFieldHandle* is NULL, the handle is not returned. If *SignatureFieldHandle* is returned, it must be released by the [SAPIHandleRelease](#) function.

### **Return Values**

If the function succeeds, the return value is SAPI\_OK.

Some possible error codes are listed in the following table.

Value	Meaning
SAPI_ERR_SESSION_IS_NULL	SAPI session handle is NULL, was not created using <a href="#">SAPIHandleAcquire</a> , or is corrupt.
SAPI_INVALID_HANDLE	The specified handle is invalid.
SAPI_ERR_NULL_VALUE_IS_NOT_ALLOWED	<i>NewSignatureFieldStruct</i> or <i>FileUNC</i> is NULL and must not be NULL
SAPI_FAILED_TO_ALLOCATE_MEMORY	CoSign Signature Local failed to allocate the necessary memory for the new Signature Field Handle. This might be due to a shortage in system resources.
SAPI_ERR_FAILED_TO_CREATE_SIGNATURE_FIELD	A general error occurred while creating the new Signature Field. Call <a href="#">SAPIExtendedLastErrorGet</a> for more information about the exact problem.

### **Remarks**

By default, you cannot create a signature field in a Word XP/2003 file using CoSign Signature Local. It is recommended to create signature fields using the Word application and the ARX Legacy

Word plug-in, and use CoSign Signature Local only for signing. If CoSign Signature Local must be used to create signature fields, contact ARX support for instructions on how to enable this feature.

This is relevant only for Entire File signatures.

In the case of ARX Legacy signatures for Word/Excel XP/2003, application-based signatures are not supported using CoSign Signature Local. Therefore, only Word XP/2003 Entire File signatures are supported using CoSign Signature Local.

Starting from CoSign version 6, you can create a signature field in an Office 2007/2010/2013 file using CoSign Signature Local. The newly created field can only appear on the first page or last page of the document. To add a signature field on the last page of the document, specify page number -1.

To add a signature field on the last page of a PDF document, specify page number -1. To add a signature field on the penultimate page of the document, specify page number -2, and so on.

For PDF files, use the AR\_PDF\_FLAG\_FIELD\_NAME\_SET flag. When this flag is set, the application can set the name of the created signature field. The desired name should be entered in the *Name* field in the *NewSignatureFieldStruct* parameter.

In TIFF files, only the first visible signature field is displayed.

You cannot create an empty signature field in an XML file. Use the [SAPISignatureFieldCreateSign](#) function for creating and signing the XML file.

***See Also***

[SAPIHandleAcquire](#), [SAPIHandleRelease](#)

## SAPISignatureFieldCreate

The **SAPISignatureFieldCreate** function is the older version of the [SAPISignatureFieldCreateEx](#) function. In the **SAPISignatureFieldCreate** function, a File Handle type is not defined.

```
SAPI int SAPISignatureFieldCreate (
    SAPI_SES_HANDLE           Handle,
    SAPI_ENUM_FILE_TYPE      FileType,
    SAPI_LPCWSTR              FileUNC,
    PSAPI_SIG_FIELD_SETTINGS NewSignatureFieldStruct,
    unsigned long             Flags,
    SAPI_SIG_FIELD_HANDLE     *SignatureFieldHandle
);
```

## SAPISignatureFieldLocatorEnumInit

The **SAPISignatureFieldLocatorEnumInit** function initializes the operation of enumerating the field locator strings inside a PDF document. These field locator strings can subsequently be used to automate the creation of new signature fields using the [SAPISignatureFieldCreate](#) function.

Note that the **SAPISignatureFieldLocatorEnumInit** function is currently supported only for PDF files.

When preparing a file by inserting field locator strings into the desired locations, keep in mind the following:

Make sure to use the same opening pattern (such as “<<<”) and closing pattern (such as “>>>”) in all field locator strings.

You can use the following string directives between the opening pattern and closing pattern:

- ◆ “W=number” or “w=number” – the width of the intended signature field
- ◆ “H=number” or “h=number” – the height of the intended signature field
- ◆ “N=text” or “n=text” – the name of the intended signature field
- ◆ “F=integer” or “f=integer” – the value of the intended signature field’s flags. When not specified, the default value is AR\_PDF\_DISABLE\_REASON (1). For the full list of possible values, refer to [Flags when Using a Signature Field in PDF Documents](#).
- ◆ “D=text” or “d=text” – the date format of the intended signature field. When not specified, the default value is “MMM d yyyy”. For the full list of possible values, refer to [DateFormat](#).
- ◆ “T=text” or “t=text” – the time format of the intended signature field. When not specified the default value is “HH:mm”. For the full list of possible values, refer to [TimeFormat](#).
- ◆ “E=text” or “e=text” – the extended time format of the intended signature field. For the full list of possible values, refer to the [SAPI\\_ENUM\\_EXTENDED\\_TIME\\_FORMAT](#) enum values. When not specified, the default value is [SAPI\\_ENUM\\_EXTENDED\\_TIME\\_FORMAT GMT](#).
- ◆ “A=number” or “a=number” – the appearance mask of the intended signature field. For more information about the appearance mask, refer to [SAPISignatureFieldCreateEx](#).  
When not specified, the default value is  
SAPI\_ENUM\_DRAWING\_ELEMENT\_GRAPHICAL\_IMAGE |  
SAPI\_ENUM\_DRAWING\_ELEMENT\_SIGNED\_BY |  
SAPI\_ENUM\_DRAWING\_ELEMENT\_TIME (i.e., 1+2+8 == 11)

The string directives must be separated using semi-colons. In addition, insert a semi-colon just before the closing pattern.

The **SAPISignatureFieldLocatorEnumInit** function and its subsequent [SAPISignatureFieldLocatorEnumCont](#) function do not delete the field locator strings. It is therefore recommended that the field locator strings be hidden.

If the PDF file was created by converting a file of a different type to PDF, you can insert the field locator strings in the original file on condition that the strings will remain intact when the file is converted to PDF.

The following is an example of a field locator string: “<<<W=100;H=120;N=employee;A=1;>>>”

```
SAPI int SAPISignatureFieldLocatorEnumInit (  
    SAPI_SES_HANDLE          Handle,  
    SAPI_CONTEXT             *SigFieldLocatorContext,  
    SAPI_FILE_HANDLE         FileHandle,
```

```
SAPI_LPCWSTR           OpeningPattern,  
PSAPI_SIG_FIELD_SETTINGS ClosingPattern,  
unsigned long          Flags,  
long                   *SignatureFieldLocateNum  
);
```

### ***Parameters***

#### *Handle*

[in] Handle of the SAPI session created by using [SAPIHandleAcquire](#).

#### *SigFieldLocatorContext*

[out] Pointer to the signature field locator enumeration operation context. This context contains internal information about the operation. The information in this context is passed in all subsequent calls to [SAPISignatureFieldLocatorEnumCont](#).

The application may not change the value of *SigFieldLocatorContext*; doing so may lead to unexpected behavior.

*SigFieldLocatorContext* must be released by the

SAPIContextRelease function when there is no further need to call the [SAPISignatureFieldLocatorEnumCont](#) function.

### *FileHandle*

[in] File handle. The SAPI file handle to be used for accessing the PDF file content.

### *Opening pattern*

[in] The pattern that indicates the start of the embedded field locator string. All given parameters will be located right after the opening pattern.

### *Closing pattern*

[in] The pattern that indicates the end point of the embedded field locator string. This parameter can be NULL. When it is NULL, the retrieved location information will include only the position of the signature field (X,Y) but will not include any other information such as Width or Height.

In cases where the orientation of the field locator string differs from the orientation of the page, this parameter must be NULL.

### *Flags*

[in] Flag values bit mask. If this value is 0, the function assumes that the text direction is from the left to right. If the flag contains the value AR\_LOCATOR\_TEXT\_DIRECTION\_AS\_PAGE\_ROTATION, the function assumes that the orientation of the field locator string is the same as the orientation of the page.

### *SignatureFieldLocatedNum*

[out] The number of markers that are found in the PDF document.

## **Return Values**

If the function succeeds, the return value is SAPI\_OK.

Some possible error codes are listed in the following table.

Value	Meaning
SAPI_ERR_SESSION_IS_NULL	SAPI session handle is NULL, was not created using <a href="#">SAPIHandleAcquire</a> , or is corrupt.
SAPI_INVALID_HANDLE	The specified handle is invalid.
SAPI_ERR_NULL_VALUE_IS_NOT_ALLOWED	<i>SigFieldLocatorContext</i> or <i>FileHandle</i> or <i>OpeningPattern</i> are NULL.
SAPI_ERR_INVALID_HANDLE	The handle that was passed to the function is not valid.
SAPI_ERR_FILE_TYPE_NOT_SUPPORTED	Currently only PDF files are supported.
SAPI_PDF_ERR_NO_DOC_PERMISSION	The permissions of the file that is processed do not allow performing the requested operation.
SAPI_ERR_LOCATOR_INVALID_CONTEXT	Invalid context handed to enum field locators.

Value	Meaning
SAPI_ERR_LOCATOR_COULD_NOT_INIT_FILE	Failed to parse the file as PDF file. This can occur due to a wrong file name, the file is missing, or in cases where the file is encrypted the failure can occur if the owner password and the user password are wrong.

**Remarks**

The following is relevant if CoSign Signature Local is accessing a file that was saved on the disk (rather than a file residing in a memory buffer): After calling the **SAPISignatureFieldLocatorEnumInit** function and until the last call to [SAPISignatureFieldEnumCont](#), or until the last usage of one of the signature fields' handles, it is recommended not to change the file using an application other than CoSign Signature Local. Doing so may lead to unexpected behavior.

The Closing Pattern must be NULL if the orientation of the field locator string differs from the orientation of the page.

If the file is protected, you will need to set either SAPI\_ENUM\_CONF\_ID\_PDF\_OWNER\_PWD or SAPI\_ENUM\_CONF\_ID\_USER\_PWD values prior to calling this function.

**See Also**

[SAPISignatureFieldLocatorEnumCont](#), [SAPIHandleRelease](#)

## SAPISignatureFieldLocatorEnumCont

The **SAPISignatureFieldLocatorEnumCont** is used for getting the [SAPI SIG FIELD SETTINGS](#) structure as well as the field locator string that was embedded in the file, for each field locator string. This information can be used to create new signature fields inside the PDF document using the [SAPISignatureFieldCreate](#) function.

Note that the **SAPISignatureFieldLocatorEnumCont** function is currently supported only for PDF files.

```
SAPI int SAPISignatureFieldLocatorEnumCont (
    SAPI_SES_HANDLE          Handle,
    SAPI_CONTEXT             *SigFieldLocatorContext,
    SAPI_SIG_FIELD_SETTINGS *SigFieldSettings,
    SAPI_LPWSTR              EncodedMessage,
    long                     *EncodedMessageLength
);
```

### **Parameters**

*Handle*

[in] Handle of the SAPI session created by using [SAPIHandleAcquire](#).

*SigFieldLocatorContext*

[in] Pointer to the signature field locator enumeration operation context created using [SAPISignatureFieldLocatorEnumInit](#).

*SigFieldLocatorContext* must be released by the

SAPIContextRelease function when there is no further need to call the [SAPISignatureFieldLocatorEnumCont](#) function.

### *SigFieldSetting*

[out] Pointer to the [SAPI SIG FIELD SETTINGS](#) structure. This structure contains all signature field information that can be used by the [SAPISignatureFieldCreate](#) function to create a new signature field. *SigFieldSetting* must be allocated (either statically or dynamically) by the application. Only the structure's fields are allocated by the **SAPISignatureFieldLocatorEnumCont** function and are released by calling the [SAPIStructRelease](#) function.

### *EncodedMessage*

[out] The text that is contained inside the opening and closing patterns. If this parameter is NULL, the EncodedMessage parameter is not retrieved.

### *EncodedMessageLen*

[in/out] The length of the EncodedMessage.

### **Return Values**

If the function succeeds, the return value is SAPI\_OK.

Some possible error codes are listed in the following table.

<b>Value</b>	<b>Meaning</b>
SAPI_ERR_SESSION_IS_NULL	SAPI session handle is NULL, was not created using <a href="#">SAPIHandleAcquire</a> , or is corrupt.
SAPI_ERR_NULL_VALUE_IS_NOT_ALLOWED	<i>SigFieldLocatorContext</i> , <i>SigFieldSettings</i> , or <i>EncodedMessageLength</i> are NULL.
SAPI_ERROR_NO_MORE_ITEMS	Reached the last located item.

### **See Also**

[SAPISignatureFieldLocatorEnumInit](#), [SAPISignatureFieldCreate](#), [SAPIHandleRelease](#)

## SAPISignatureFieldSignEx

The **SAPISignatureFieldSignEx** function signs a specific signature field in a specific file. The file name and type, as well as the attributes for the signing operation, are all derived from the *SignatureFieldHandle*. Use this function also when the *prompt for sign* feature is enabled.

```
SAPI int SAPISignatureFieldSignEx (
    SAPI_SES_HANDLE          Handle,
    SAPI_SIG_FIELD_HANDLE    SignatureFieldHandle,
    unsigned long            Flags,
    unsigned char            *Credential,
    unsigned long            CredentialLen
);
```

### Parameters

#### Handle

[in] Handle of the SAPI session created by using [SAPIHandleAcquire](#).

#### SignatureFieldHandle

[in] Pointer to the Signature Field handle that represents the field to be signed. The Signature Field handle was created by either the [SAPISignatureFieldEnumCont](#) function or the [SAPISignatureFieldCreate](#) function.

*SignatureFieldHandle* must be released by the [SAPIHandleRelease](#) function.

#### Flags

[in] Flag values bit mask. For more information, refer to the following sections in Appendix D: [Flags when Signing Word Documents](#), [Flags when Signing PDF Files](#), [Flags when Signing TIFF Files](#), and [Time-Stamping/OCSP and Miscellaneous Signature-Related Flags](#).

#### Credential

[in] Pointer to a wide char buffer that holds the user credentials required for the signing operation. Contact ARX support for more information. If no special credential is needed, a NULL value can be supplied. However, if the *prompt for sign* feature is enabled, you must supply credentials.

#### CredentialLen

[in] The length in bytes of the content of *Credential*, including NULL separators and/or terminators. If no special credentials are needed, a value of 0 can be supplied.

### Return Values

If the function succeeds, the return value is SAPI\_OK.

Some possible error codes are listed in the following table.

Value	Meaning
SAPI_ERR_SESSION_IS_NULL	SAPI session handle is NULL, was not created using <a href="#">SAPIHandleAcquire</a> , or is corrupt.
SAPI_ERR_NULL_VALUE_IS_NOT_ALLOWED	<i>SignatureFieldHandle</i> is NULL and must not be NULL.

Value	Meaning
SAPI_ERR_NO_CERT_TO_SELECT_FROM	No certificate was found. This might be due to incorrect credentials, networking problems, or because the preferred certificate could not be found.
SAPI_ERR_TOO_MANY_CERTS_TO_SELECT_FROM	More than one certificate is available, and CoSign Signature Local cannot choose between them. Use either <a href="#">SAPICertificateSetDefault</a> or <a href="#">SAPIConfigurationValueSet</a> to set the certificate with which CoSign Signature Local works.
SAPI_ERR_FAILED_TO_SIGN_FILE	A general error occurred while signing the file. Call <a href="#">SAPIExtendedLastErrorGet</a> for more information about the exact problem.

### Remarks

The certificate that the **SAPISignatureFieldSignEx** function uses for signing is either the one set as the default certificate (for example, by calling the [SAPICertificateSetDefault](#) function), or a certificate that the function automatically chooses. See the [SAPICertificateGetDefault](#) function for more information about the automatic algorithm for choosing a certificate.

To improve performance, if no default certificate is set prior to the signing operation, CoSign Signature Local stores the certificate it used for this operation as the default for the purpose of certifying the next signing operation in the current SAPI session. If this behavior is not required, set the configuration value of [SAPI\\_ENUM\\_CONF\\_ID\\_CERT\\_SET\\_DEFAULT](#) to 0 before the initial call to **SAPISignatureFieldSignEx** in the current session.

- If a graphical image is required by the signature field attribute and there is only one graphical signature stored in DocuSign Signature Appliance, it is retrieved automatically by the **SAPISignatureFieldSignEx** function. If more than one graphical signature is stored in DocuSign Signature Appliance, use [SAPIGraphicSigImageSetDefaultEx](#) to set a default image, or use [SAPIConfigurationValueSet](#) with the *SAPI\_ENUM\_CONF\_ID\_GR\_SIG\_PREF\_NAME* parameter to set the graphical image to use.
- If a logo is required by the signature field attribute and there is only one graphical signature stored in DocuSign Signature Appliance, it is retrieved automatically by the **SAPISignatureFieldSignEx** function. If more than one logo is stored in DocuSign Signature Appliance, use [SAPIGraphicSigImageSetDefaultEx](#) to set a default logo, or use [SAPIConfigurationValueSet](#) with the *SAPI\_ENUM\_CONF\_ID\_LOGO\_PREF\_NAME* parameter to set the logo to use.  
Note that starting from version 5.4, only one logo is permissible for a given account.
- If an initials image is required by the signature field attribute and there is only one graphical signature stored in DocuSign Signature Appliance, it is retrieved automatically by the **SAPISignatureFieldSignEx** function. If more than one initials image is stored in DocuSign Signature Appliance, use [SAPIGraphicSigImageSetDefaultEx](#) to set a default initials image. If a reason is attached to the signature field, it must be set as a configuration value using the [SAPIConfigurationValueSet](#) function prior to calling the **SAPISignatureFieldSignEx** function.
- If a title is attached to the signature field, it must be set as a configuration value using the [SAPIConfigurationValueSet](#) function prior to calling the **SAPISignatureFieldSignEx** function.

- In the case of ARX Legacy signatures for Word/Excel XP/2003, application-based signatures are not supported using CoSign Signature Local. Therefore, only Word XP/2003 Entire File signatures are supported using CoSign Signature Local.
- In the case of PDF files, the maximum allowed sizes of Graphical signatures, Logo or Initials is one megabyte.

***See Also***

[SAPIHandleAcquire](#), [SAPICertificateSetDefault](#), [SAPICertificateGetDefault](#), [SAPIConfigurationValueSet](#), [SAPIHandleRelease](#), [SAPISignatureFieldCreate](#), [SAPISignatureFieldEnumCont](#), [SAPIGraphicSigImageSetDefault](#), [SAPISignatureFieldSign](#)

## SAPISignatureFieldSign

The **SAPISignatureFieldSign** function is the older version of the [SAPISignatureFieldSignEx](#) function.

```
SAPI int SAPISignatureFieldSign (  
    SAPI_SES_HANDLE          Handle,  
    SAPI_SIG_FIELD_HANDLE    SignatureFieldHandle,  
    unsigned long            Flags  
);
```

## SAPISignatureFieldCreateSignEx

The **SAPISignatureFieldCreateSignEx** function both creates a new field and signs that field. This function, created for convenience, internally calls the [SAPISignatureFieldCreateEx](#) and [SAPISignatureFieldSignEx](#) functions. For more information, refer to the descriptions of these two functions.

```
SAPI int SAPISignatureFieldCreateSignEx (  
    SAPI_SES_HANDLE           Handle,  
    SAPI_ENUM_FILE_TYPE      FileType,  
    SAPI_LPCWSTR              FileUNC,  
    SAPI_FILE_HANDLE         FileHandle,  
    PSAPI_SIG_FIELD_SETTINGS NewSignatureFieldStruct,  
    unsigned long             Flags,  
    unsigned char             *Credential,  
    unsigned long             CredentialLen);
```

### **Parameters**

#### *Handle*

[in] Handle of the SAPI session created by using [SAPIHandleAcquire](#).

#### *FileType*

[in] File type.

#### *FileUNC*

[in] File name. It is recommended to use the *FileHandle* parameter instead and provide "" as an input parameter.

#### *FileHandle*

[in] File handle. The SAPI file handle to be used for the field creation and signature operation. The file handle can be created using either the [SAPICreateFileHandleByName](#) function or the

SAPICreateFileHandleByMem function. If this value is NULL, the *FileUNC* parameter is used.

### *NewSignatureFieldStruct*

[in] Pointer to the [SAPI\\_SIG\\_FIELD\\_SETTINGS](#) structure. This structure holds all the non-crypto information of the newly created signature field.

### *Flags*

[in] Flag values bit mask. For more information, refer to the following sections in Appendix D: [Flags when Creating a Signature Field in PDF Files](#), [Flags when Signing PDF Files](#), [Flags when Creating a Signature Field in Office 2007/2010/2013 Documents](#), [Flags when Signing XML Documents](#), and [Time-Stamping/OCSP and Miscellaneous Signature-Related Flags](#).

Note that some of the signature field flags are passed as part of the function call, while others are passed as part of the signature field [SAPI\\_SIG\\_FIELD\\_SETTINGS](#) structure to which the function points. The relevant sections in [Appendix D](#) note which flags are passed as part of the function and which are passed as part of the signature field structure.

### *Credential*

[in] Pointer to a wide char buffer that holds the user credentials required for the signing operation. Contact ARX support for more information. If no special credential is needed, a NULL value can be supplied. However, if the *prompt for sign* feature is enabled, you must supply credentials.

### *CredentialLen*

[in] The length in bytes of the content of *Credentials* including NULL separators and/or terminators. If no special credential is needed, a value of 0 can be supplied.

### **Return Values**

If the function succeeds, the return value is SAPI\_OK.

Some possible error codes are listed in the following table.

Value	Meaning
SAPI_ERR_SESSION_IS_NULL	SAPI session handle is NULL, was not created using <a href="#">SAPIHandleAcquire</a> , or is corrupt.
SAPI_INVALID_HANDLE	The specified handle is invalid.
SAPI_ERR_NULL_VALUE_IS_NOT_ALLOWED	<i>SignatureFieldHandle</i> is NULL and must not be NULL.
SAPI_ERR_NO_CERT_TO_SELECT_FROM	No certificate was found. This might be due to incorrect credentials, networking problems, or because the preferred certificate could not be found.
SAPI_ERR_TOO_MANY_CERTS_TO_SELECT_FROM	More than one certificate is available, and CoSign Signature Local cannot choose between them. Use either <a href="#">SAPICertificateSetDefault</a> or <a href="#">SAPIConfigurationValueSet</a> to set the certificate with which CoSign Signature Local operates.

Value	Meaning
SAPI_ERR_FAILED_TO_SIGN_FILE	A general error occurred while signing the file. Call <a href="#">SAPIExtendedLastErrorGet</a> for more information about the exact problem.

**Remarks**

See the description of the two base functions: [SAPISignatureFieldCreate](#) and [SAPISignatureFieldSignEx](#). Any limitations that apply to [SAPISignatureFieldCreate](#) and/or [SAPISignatureFieldSignEx](#), apply to this function as well. Therefore, carefully read the description of these two functions.

This function is relevant only for PDF and XML file types. If you wish to create and sign new signature fields in other file types, you must call two separate functions: [SAPISignatureFieldCreate](#) for creating a signature field, and [SAPISignatureFieldSignEx](#) for signing the newly created signature fields.

**See Also**

[SAPIHandleAcquire](#), [SAPICertificateSetDefault](#), [SAPICertificateGetDefault](#), [SAPIConfigurationValueSet](#), [SAPIHandleRelease](#), [SAPISignatureFieldCreate](#), [SAPISignatureFieldEnumCont](#), [SAPIGraphicSigImageSetDefault](#), [SAPISignatureFieldSignEx](#)

## **SAPISignatureFieldCreateSign**

The **SAPISignatureFieldCreateSign** function is the older version of the

SAPISignatureFieldCreateSignEx function. In the **SAPISignatureFieldCreateSign** function, a File Handle type is not defined.

```
SAPI int SAPISignatureFieldCreateSign (  
    SAPI_SES_HANDLE           Handle,  
    SAPI_ENUM_FILE_TYPE      FileType,  
    SAPI_LPCWSTR              FileUNC,  
    PSAPI_SIG_FIELD_SETTINGS NewSignatureFieldStruct,  
    unsigned long             Flags,  
    unsigned char             *Credential,  
    unsigned long             CredentialLen);
```

## SAPISignatureFieldVerify

The **SAPISignatureFieldVerify** function verifies the validity of a single signature field. The function returns `SAPI_OK` if the signature is valid. If the signature is valid, it also checks the certificate status, and returns its value in *CertificateStatus*.

```
SAPI int SAPISignatureFieldVerify (
    SAPI_SES_HANDLE          Handle,
    SAPI_SIG_FIELD_HANDLE    SignatureFieldHandle,
    SAPI_CERT_STATUS_STRUCT  *CertificateStatus,
    unsigned long            Flags
);
```

### Parameters

#### Handle

[in] Handle of the SAPI session created by using [SAPIHandleAcquire](#).

#### SignatureFieldHandle

[in] Pointer to the Signature Field handle that represents the field to be verified. The Signature Field handle was created by the [SAPISignatureFieldEnumCont](#) function.

*SignatureFieldHandle* must be released by the [SAPIHandleRelease](#) function.

#### CertificateStatus

[out] The validity status of the certificate. This parameter must be supplied.

#### Flags

[in] Flag values bit mask. For more information, refer to the following section in Appendix D: [Flags when Verifying a Signature](#).

### Return Values

If the function succeeds, the return value is `SAPI_OK`.

Some possible error codes are listed in the following table.

Value	Meaning
<code>SAPI_ERR_SESSION_IS_NULL</code>	SAPI session handle is NULL, was not created using <a href="#">SAPIHandleAcquire</a> , or is corrupt.
<code>SAPI_ERR_NULL_VALUE_IS_NOT_ALLOWED</code>	<i>SignatureFieldHandle</i> is NULL and must not be NULL.
<code>SAPI_ERR_FAILED_TO_VERIFY_FILE</code>	A general error occurred while verifying a signature. Call <a href="#">SAPIExtendedLastErrorGet</a> for more information about the exact problem.

### Remarks

It is recommended to carefully check the certificate status and make sure its value is as expected. For example, whereas in some environments the inability to check a CRL can be a real problem, in other environments it cannot be checked at all.

To control the attributes to be checked when verifying a certificate, set `SAPI_ENUM_CONF_ID_CERT_CHAIN_FLAGS` to a desired value (see

[SAPI\\_ENUM\\_CONF\\_ID](#)), using the [SAPIConfigurationValueSet](#) function. Contact ARX for information about which value to pass.

To check if the certificate is revoked, set `SAPI_ENUM_CONF_ID_CHK_CRL_VERIFY` or `SAPI_ENUM_CONF_ID_CHK_CRL_ENUM` (see [SAPI\\_ENUM\\_CONF\\_ID](#)) to `True`, using the [SAPIConfigurationValueSet](#) function.

When verifying a signed file, you can use the [SAPIFileIsSigned](#) function before enumerating the signature fields, to ensure that the file is signed (i.e., there is at least one signed signature field).

If time-stamping is used, the returned certificate status will be based on the status of both the user certificate as well as the time-stamping server certificate.

If the user's certificate is not valid, this will be reflected in the certificate status.

If the user's certificate is valid, then the time-stamping certificate status will be retrieved. A

`SAPI_ENUM_CERT_STATUS_OK` means that both the user and the time-stamping server certificates are OK.

To retrieve the exact time-stamping certificate status, use the [SAPI\\_ENUM\\_CONF\\_ID\\_SECURED\\_TS\\_CERT\\_STATUS](#) configuration value retrieval.

### ***See Also***

[SAPIHandleAcquire](#), [SAPISignatureFieldSign](#), [SAPIHandleRelease](#), [SAPISignatureFieldEnumCont](#), [SAPIConfigurationValueSet](#), [SAPIFileIsSigned](#)

## SAPIFileIsSignedEx

The **SAPIFileIsSignedEx** function returns true in *IsSigned* if *FileUNC* has at least one signed signature field. Otherwise, *IsSigned* is false.

This function is useful for applications that display a special icon for signed files, or that enable the Verify menu option upon right-clicking the file name, etc.

```
SAPI int SAPIFileIsSignedEx (  
    SAPI_SES_HANDLE          Handle,  
    SAPI_ENUM_FILE_TYPE     FileType,  
    SAPI_LPCWSTR             FileUNC,  
    SAPI_FILE_HANDLE        FileHandle,  
    unsigned long            Flags;  
    SAPI_BOOL                *IsSigned  
);
```

### Parameters

#### *Handle*

[in] Handle of the SAPI session created by using [SAPIHandleAcquire](#).

#### *FileType*

[in] File type. Refer to [SAPI\\_ENUM\\_FILE\\_TYPE](#).

#### *FileUNC*

[in] File name. It is recommended to use the *FileHandle* parameter instead and provide "" as an input parameter.

#### *FileHandle*

[in] File handle. The file handle can be created using either the [SAPICreateFileHandleByName](#) function or the

SAPICreateFileHandleByMem function. If this value is NULL, the *FileUNC* parameter is used.

### *Flags*

[in] Reserved for future use and must be zero.

### *IsSigned*

[in] A pointer to a flag specifying whether the file has at least one signed signature field (true), or not (false).

The supplied flag must be set to 0 prior to calling the API call.

### **Return Values**

If the function succeeds, the return value is `SAPI_OK`.

Some possible error codes are listed in the following table.

Value	Meaning
<code>SAPI_ERR_SESSION_IS_NULL</code>	SAPI session handle is NULL, was not created using <a href="#">SAPIHandleAcquire</a> , or is corrupt.
<code>SAPI_INVALID_HANDLE</code>	The specified handle is invalid.
<code>SAPI_ERR_FILE_TYPE_NOT_SUPPORTED</code>	The file type provided in <i>FileType</i> is not one of the supported file types.
<code>SAPI_ERR_NULL_VALUE_IS_NOT_ALLOWED</code>	<i>FileUNC</i> or <i>IsSigned</i> is NULL and must not be NULL.
<code>SAPI_ERR_FAILED_IN_IS_FILE_SIGNED</code>	A general error occurred while checking whether the file is signed. Call <a href="#">SAPIExtendedLastErrorGet</a> for more information about the exact problem.

### **Remarks**

This function does not verify the signature fields in the file, it only indicates whether there are signed signature fields in the file.

### **See Also**

[SAPIHandleAcquire](#), [SAPISignatureFieldSign](#), [SAPISignatureFieldVerify](#)

## SAPIFileIsSigned

The **SAPIFileIsSigned** function is the older version of the [SAPIFileIsSignedEx](#) function. In the **SAPIFileIsSigned** function, a File Handle type is not defined.

```
SAPI int SAPIFileIsSigned (
    SAPI_SES_HANDLE          Handle,
    SAPI_ENUM_FILE_TYPE     FileType,
    SAPI_LPCWSTR             FileUNC,
    unsigned long            Flags;
    SAPI_BOOL                *IsSigned
);
```

## SAPISignatureFieldClear

The **SAPISignatureFieldClear** function clears a signature field. This means that all the cryptographic-related information is cleared as well as its visual objects, but the field and its attributes remain unchanged.

```
SAPI int SAPISignatureFieldClear (
    SAPI_SES_HANDLE          Handle,
    SAPI_SIG_FIELD_HANDLE    SignatureFieldHandle
    unsigned long            Flags
);
```

### Parameters

#### Handle

[in] Handle of the SAPI session created by using [SAPIHandleAcquire](#).

#### SignatureFieldHandle

[in] Pointer to the Signature Field handle that represents the field to be cleared. The Signature Field handle was created by either the [SAPISignatureFieldEnumCont](#) function or the [SAPISignatureFieldCreate](#) function.

*SignatureFieldHandle* must be released by the [SAPIHandleRelease](#) function.

#### Flags

[in] Flag values bit mask. The available values are:

AR\_WORD\_CLEAR\_DEPENDENT\_SIGNATURES\_FLAG and

AR\_WORD\_CLEAR\_ALL\_SIGNATURES\_FLAG for Word files (see [Flags when Clearing or Removing Signature Fields](#) for more information).

### Return Values

If the function succeeds, the return value is SAPI\_OK.

Some possible error codes are listed in the following table.

Value	Meaning
SAPI_ERR_SESSION_IS_NULL	SAPI session handle is NULL, was not created using <a href="#">SAPIHandleAcquire</a> , or is corrupt.
SAPI_ERR_NULL_VALUE_IS_NOT_ALLOWED	<i>SignatureFieldHandle</i> is NULL and must not be NULL.
SAPI_ERR_FAILED_TO_CLEAR_SIGNATURE_FIELD	A general error occurred while clearing the Signature Field. Call <a href="#">SAPIExtendedLastErrorGet</a> for more information about the exact problem.

### Remarks

A signature field that was cleared can be signed again using the same Signature Field handle.

### See Also

[SAPIHandleAcquire](#), [SAPISignatureFieldEnumCont](#), [SAPISignatureFieldCreate](#), [SAPIHandleRelease](#)

## SAPISignatureFieldRemove

The **SAPISignatureFieldRemove** function removes a signature field from the file.

```
SAPI int SAPISignatureFieldRemove (
    SAPI_SES_HANDLE          Handle,
    SAPI_SIG_FIELD_HANDLE    SignatureFieldHandle,
    unsigned long            Flags
);
```

### Parameters

#### Handle

[in] Handle of the SAPI session created by using [SAPIHandleAcquire](#).

#### SignatureFieldHandle

[in] Pointer to the Signature Field handle that represents the field that is going to be removed. The Signature Field handle was created by either the [SAPISignatureFieldEnumCont](#) function or the [SAPISignatureFieldCreate](#) function.

*SignatureFieldHandle* must be released by the [SAPIHandleRelease](#) function.

#### Flags

[in] Flag values bit mask. The available values are:

AR\_WORD\_CLEAR\_DEPENDENT\_SIGNATURES\_FLAG and

AR\_WORD\_CLEAR\_ALL\_SIGNATURES\_FLAG for Word files (see [Flags when Clearing or Removing Signature Fields](#) for more information).

### Return Values

If the function succeeds, the return value is SAPI\_OK.

Some possible error codes are listed in the following table.

Value	Meaning
SAPI_ERR_SESSION_IS_NULL	SAPI session handle is NULL, was not created using <a href="#">SAPIHandleAcquire</a> , or is corrupt.
SAPI_ERR_NULL_VALUE_IS_NOT_ALLOWED	<i>SignatureFieldHandle</i> is NULL and must not be NULL.
SAPI_ERR_FAILED_TO_REMOVE_SIGNATURE_FIELD	A general error occurred while removing the Signature Field. Call <a href="#">SAPIExtendedLastErrorGet</a> for more information about the exact problem.

### Remarks

A signature field that was removed cannot be used. The only operation allowed for the Signature Field Handle is [SAPIHandleRelease](#).

A signature field can be removed whether it is signed or not.

By default, you cannot remove a signature field in a Word 2003 file using CoSign Signature Local. It is recommended to remove signature fields using the Word application and the ARX Legacy Word add-in, and use CoSign Signature Local only for signing or clearing fields. If CoSign

Signature Local must be used for signature fields removal, contact ARX support for instructions on how to enable this feature.

You cannot remove an Office 2007/2010/2013 signature using CoSign Signature Local. You must remove signatures using the Office 2007/2010/2013 application.

***See Also***

[SAPIHandleAcquire](#), [SAPISignatureFieldEnumCont](#), [SAPISignatureFieldCreate](#), [SAPIHandleRelease](#)

## SAPISignatureFieldDetailsGUIGet

The **SAPISignatureFieldDetailsGUIGet** displays information related to the signature status and the signing certificate.

```
SAPI int SAPISignatureFieldDetailsGUIGet (  
    SAPI_SES_HANDLE          Handle,  
    SAPI_SIG_FIELD_HANDLE    SignatureFieldHandle  
);
```

### Parameters

#### Handle

[in] Handle of the SAPI session created by using [SAPIHandleAcquire](#).

#### SignatureFieldHandle

[in] Pointer to the Signature Field handle that represents the signed field. The information that is displayed on screen is related to this signed field.

### Return Values

If the function succeeds, the return value is SAPI\_OK.

Some possible error codes are listed in the following table.

Value	Meaning
SAPI_ERR_SESSION_IS_NULL	SAPI session handle is NULL, was not created using <a href="#">SAPIHandleAcquire</a> , or is corrupt.
SAPI_ERR_NULL_VALUE_IS_NOT_ALLOWED	<i>SignatureFieldHandle</i> is NULL and must not be NULL.

### See Also

[SAPIHandleAcquire](#), [SAPISignatureFieldSign](#)

## SAPISigningCeremonyGUI

The **SAPISigningCeremonyGUI** displays the standard signing ceremony dialog.

The values that the end-user enters in the dialog are saved in CoSign Signature Local, and will appear as the default values the next time this dialog is displayed.

```
SAPI int SAPISigningCeremonyGUI(  
    SAPI_SES_HANDLE                Handle,  
    SAPI_SIG_FIELD_HANDLE          SignatureFieldHandle,  
    unsigned long                  flags,  
    unsigned long                  DisplayComponentsMask,  
    unsigned long                  GraphicFormatsPermitted,  
    SAPI_LPCWSTR                   InstructToSigner,  
    SAPI_ENUM_CEREMONY_COMPONENT_SHOW_MODE ImageSelection,  
    SAPI_ENUM_CEREMONY_COMPONENT_SHOW_MODE CertSelection,  
    SAPI_ENUM_CEREMONY_COMPONENT_SHOW_MODE ReasonSelection,  
    SAPI_ENUM_CEREMONY_COMPONENT_SHOW_MODE TitleSelection,  
    SAPI_ENUM_GR_IMG_SELECT_MODE    GraphicalSelectionMode  
);
```

### Parameters

#### Handle

[in] Handle of the SAPI session created by using [SAPIHandleAcquire](#).

#### SignatureFieldHandle

[in] Pointer to the Signature Field handle that represents the specific signature field. The specific signature field settings determine which signature components appear in the displayed signing ceremony dialog.

If NULL is passed, then the list of signature components to display is derived from the *DisplayComponentsMask*.

#### Flags

[in] Flag values bit mask. For more information, refer to the following section in Appendix D: [Flags when Using the Signing Ceremony API](#).

#### DisplayComponentsMask

[in] This parameter is relevant if *SignatureFieldHandle* is NULL. The value is built as a mask, based on the [SAPI\\_ENUM\\_DRAWING\\_ELEMENT](#) enum. In this way, you can specify the exact structure of the signing ceremony dialog.

#### GraphicFormatsPermitted

[in] Specify which types of graphical images to present for selection. You can send one of the following values:

```
AR_GR_FLAG_IMAGE_TYPE,  
AR_GR_FLAG_INITIALS_TYPE,  
AR_GR_FLAG_IMAGE_TYPE | AR_GR_FLAG_INITIALS_TYPE
```

#### InstructToSigner

[in] Displays text that provides instructions about the digital signature act.

#### ImageSelection

[in] Instructs the signing ceremony whether to include the graphical signature selection component. Use the *SAPI\_ENUM\_CEREMONY\_COMPONENT\_SHOW\_MODE\_BY\_APPEARANCE\_MASK* value.

### *CertSelection*

[in] Instructs the signing ceremony whether to include the certificate selection component. Use the SAPI\_ENUM\_CEREMONY\_COMPONENT\_SHOW\_MODE\_BY\_APPEARANCE\_MASK value.

### *ReasonSelection*

[in] Instructs the signing ceremony whether to include the reason selection component. Use the SAPI\_ENUM\_CEREMONY\_COMPONENT\_SHOW\_MODE\_BY\_APPEARANCE\_MASK value.

### *TitleSelection*

[in] Instructs the signing ceremony whether to include the title selection component. Use the SAPI\_ENUM\_CEREMONY\_COMPONENT\_SHOW\_MODE\_BY\_APPEARANCE\_MASK value.

### *GraphicalSelectionMode*

[in] Based on [SAPI\\_ENUM\\_GR\\_IMG\\_SELECT\\_MODE](#).

Use [SAPI\\_ENUM\\_GR\\_IMG\\_SEL\\_MODE\\_SELECT](#) for a regular signature ceremony procedure.

Use [SAPI\\_ENUM\\_GR\\_IMG\\_SEL\\_MODE\\_VIEW\\_USER](#) for managing a user's graphical signatures. This option is used by the graphical signatures utility in user mode.

Use [SAPI\\_ENUM\\_GR\\_IMG\\_SEL\\_MODE\\_VIEW\\_KIOSK](#) for managing a user's graphical signatures. This option is used by the graphical signatures utility in administrative mode.

To use [SAPI\\_ENUM\\_GR\\_IMG\\_SEL\\_MODE\\_VOLATILE](#), contact ARX for more information.

### ***Return Values***

If the function succeeds, the return value is SAPI\_OK.

Some possible error codes are listed in the following table.

Value	Meaning
SAPI_ERR_SESSION_IS_NULL	SAPI session handle is NULL, was not created using <a href="#">SAPIHandleAcquire</a> , or is corrupt.
SAPI_ERR_NULL_VALUE_IS_NOT_ALLOWED	<i>SignatureFieldHandle</i> is NULL and must not be NULL.

### ***See Also***

[SAPIHandleAcquire](#), [SAPISignatureFieldSign](#)

## CoSign Signature Local Certificates

This chapter describes the functions related to certificates. These functions fall into two groups:

*Retrieving certificate information* – Functions for retrieving information from a given certificate. This group includes the [SAPICertificateGetFieldByBlob](#) and [SAPIPKCS7BlobGetValue](#) functions, and in some special scenarios it includes the [SAPICertificateGetFieldByHandle](#) and [SAPICertificateGetDefault](#) functions, which are generally grouped in the second group.

The [SAPIPKCS7BlobGetValue](#) function retrieves the certificate data from a PKCS#7 formatted signature blob (either after signing data or before verifying signed data). The [SAPICertificateGetFieldByBlob](#) function retrieves various data (such as expiration dates, certificate's owner, certificate's issuer, etc.) from a specific certificate. This function can be useful in cases where an application needs to know the identity of the signer of a certain record, and it is not relevant for obtaining the identity of the signer of a file.

The [SAPICertificateGetFieldByHandle](#) function is identical to the [SAPICertificateGetFieldByBlob](#) function, except that instead of using the raw certificate data as the information source, it uses a CoSign Signature Local certificate handle. The certificate handle is returned by [SAPICertificateGetDefault](#) in a single-certificate environment, while in a multi-certificates environment you might use the enumeration functions ([SAPICertificatesEnumInit](#) and [SAPICertificatesEnumCont](#)). The [SAPICertificateGetFieldByHandle](#) function is called when the certificate information must be retrieved before the signing operation, unlike the [SAPICertificateGetFieldByBlob](#) function that is usually called after the data was signed.

*Selecting a certificate in a multi-certificate environment* – Functions for environments in which a user has multiple certificates. In most cases, users have a single certificate, and this certificate is automatically selected by CoSign Signature Local for signing. However, in multi-certificate environments, a default certificate (or its unique identifiers) must be set prior to the signing operation, otherwise CoSign Signature Local is not able to select a certificate with which to work. Use the functions in this group only if there is a possibility that more than one certificate exists.

This group of functions includes:

- ◆ The enumeration functions, [SAPICertificatesEnumInit](#) and [SAPICertificatesEnumCont](#), which return all the certificate handles one by one.
- ◆ [SAPICertificateGUISelect](#) for selecting a certificate from a list using Windows' *Select Certificate* dialog box.
- ◆ [SAPICertificateSetDefault](#) for setting a specific certificate as the default certificate for all subsequent signing operations.
- ◆ [SAPICertificateGetFieldByHandle](#) and [SAPICertificateGetDefault](#), described in the context of the first group.

## SAPICertificatesEnumInit

The **SAPICertificatesEnumInit** function initializes the certificates enumeration operation. This operation is used in environments where there might be more than one available certificate. A call to the [SAPICertificatesEnumInit](#) function is followed by a call to the [SAPICertificatesEnumCont](#) function until all the certificates are retrieved, or until the required certificate is found. To get more information about a specific certificate, call the [SAPICertificateGetFieldByHandle](#) function.

```
SAPI int SAPICertificatesEnumInit (
    SAPI_SES_HANDLE          Handle,
    SAPI_CONTEXT             *CertContext,
    unsigned long             Flags
);
```

### Parameters

#### *Handle*

[in] Handle of the SAPI session created by using [SAPIHandleAcquire](#).

#### *CertContext*

[out] Pointer to the Certificates Enumeration operation context. This context contains internal information about the operation. The information in this context is passed in all subsequent calls to [SAPICertificatesEnumCont](#). Changing the value of *CertContext* by the application is not allowed and can lead to unexpected behavior.

*CertContext* must be released by the

SAPIContextRelease function when the certificates enumeration operation is finished.

### Flags

[in] Flag values bit mask. For more information, refer to the following section in Appendix D: [Flags when Enumerating Certificates](#).

### Return Values

If the function succeeds, the return value is SAPI\_OK.

Some possible error codes are listed in the following table.

Value	Meaning
SAPI_ERR_SESSION_IS_NULL	SAPI session handle is NULL, was not created using <a href="#">SAPIHandleAcquire</a> , or is corrupt.
SAPI_ERR_NULL_VALUE_IS_NOT_ALLOWED	<i>CertContext</i> is NULL and must not be NULL.
SAPI_FAILED_TO_ALLOCATE_MEMORY	CoSign Signature Local failed to allocate the necessary memory for the new operation context. This might be due to a shortage in system resources.
SAPI_ERR_FAILED_TO_INIT_CERT_ENUM	A general error occurred while initializing the certificates enumeration operation. Call <a href="#">SAPIExtendedLastErrorGet</a> for more information about the exact problem.

### Remarks

**SAPICertificatesEnumInit** builds an internal list of certificates according to the available certificates at the moment it is called. Only certificates from the list are returned by the [SAPICertificatesEnumCont](#) function. Therefore, new certificates that become available after the **SAPICertificatesEnumInit** returns, do not affect the certificate handles that are retrieved by the [SAPICertificatesEnumCont](#) function. However, removing certificates after **SAPICertificatesEnumInit** returns, can cause [SAPICertificatesEnumCont](#) to fail.

In environments in which there is only one certificate, or when the required certificate can be identified by its serial ID or by its public key's hash value (Subject Key Identifier), the certificates enumeration operation is unnecessary.

Use the [PERFORM\\_EXTENDED\\_VALIDATION](#) constant to enforce the extended certificate path validation tests. Only certificates that pass the validation tests can be used for signature operations.

### See Also

[SAPIHandleAcquire](#),

SAPIContextRelease, [SAPICertificatesEnumCont](#), [SAPICertificateGetFieldByHandle](#),  
[SAPICertificateGetDefault](#)

## SAPICertificatesEnumCont

The **SAPICertificatesEnumCont** function continues the certificates enumeration operation and returns a pointer to a single certificate handle from the internal certificates list built by the [SAPICertificatesEnumInit](#) function. Every call returns a new certificate handle until the end of the list is reached.

```
SAPI int SAPICertificatesEnumCont (
    SAPI_SES_HANDLE          Handle,
    SAPI_CONTEXT             *CertContext,
    SAPI_CERT_HANDLE         *CertHandle
);
```

### ***Parameters***

#### *Handle*

[in] Handle of the SAPI session created by using [SAPIHandleAcquire](#).

#### *CertContext*

[in] Pointer to the Certificate Enumeration operation context created using [SAPICertificatesEnumInit](#). *CertContext* must be released by the

SAPIContextRelease function when there are no more certificates to retrieve or when the required certificate was found.

### *CertHandle*

[out] Pointer to a handle of a certificate. This handle can be used when calling other certificate functions, such as [SAPICertificateGetFieldByHandle](#) and [SAPICertificateSetDefault](#).

When *CertHandle* is not in use anymore, it is released by calling the [SAPIHandleRelease](#) function.

### **Return Values**

If the function succeeds, the return value is SAPI\_OK.

Some possible error codes are listed in the following table.

Value	Meaning
SAPI_ERR_SESSION_IS_NULL	SAPI session handle is NULL, was not created using <a href="#">SAPIHandleAcquire</a> , or is corrupt.
SAPI_ERR_NULL_VALUE_IS_NOT_ALLOWED	<i>CertContext</i> or <i>CertHandle</i> is NULL and must not be NULL.
SAPI_FAILED_TO_ALLOCATE_MEMORY	CoSign Signature Local failed to allocate the necessary memory for the certificate handle. This might be due to a shortage in system resources.
SAPI_ERR_CONTEXT_NOT_INITIALIZED	The certificates enumeration context is either not initialized or is corrupt.
SAPI_ERROR_NO_MORE_ITEMS	There are no more certificates to retrieve.
SAPI_ERR_FAILED_TO_CONT_CERT_ENUM	A general error occurred while continuing the certificates enumeration operation. Call <a href="#">SAPIExtendedLastErrorGet</a> for more information about the exact problem.

### **Remarks**

**SAPICertificatesEnumCont** returns only the certificates that were available when [SAPICertificatesEnumInit](#) was called.

In order to retrieve the whole certificate list, **SAPICertificatesEnumCont** is called until SAPI\_ERROR\_NO\_MORE\_ITEMS is returned.

### **See Also**

[SAPIHandleAcquire](#),

SAPIContextRelease, [SAPICertificatesEnumInit](#), [SAPICertificateGetFieldByHandle](#),  
[SAPICertificateSetDefault](#)

## SAPICertificateGetFieldByHandle

The **SAPICertificateGetFieldByHandle** function retrieves a single field from a certificate pointed to by a certificate handle.

```
SAPI int SAPICertificateGetFieldByHandle (
    SAPI_SES_HANDLE          Handle,
    SAPI_CERT_HANDLE        CertHandle,
    SAPI_ENUM_CERT_FIELD    FieldID,
    unsigned char            *Value,
    unsigned long            *ValueLen,
    SAPI_ENUM_DATA_TYPE     *FieldType
);
```

### Parameters

#### Handle

[in] Handle of the SAPI session created by [SAPIHandleAcquire](#).

#### CertHandle

[in] Handle of a certificate created using [SAPICertificateGetDefault](#) or [SAPICertificatesEnumCont](#).

#### FieldID

[in] The ID of the certificate field to be retrieved. A description of all available IDs is contained in SAPI\_ENUM\_CERT\_FIELD.

#### Value

[out] Pointer to a buffer holding the data retrieved according to the value of *FieldID*. This parameter can be NULL if only the value length is required.

#### ValueLen

[in/out] Pointer to a value that specifies the size allocated for the *Value* parameter. When the function returns, this value contains the size of the data copied to *Value*, or if *Value* is NULL, the size that needs to be allocated for *Value*.

#### FieldType

[out] Pointer to a value that receives a code indicating the type of data that was returned in *Value*. For a list of the possible type codes, see [SAPI\\_ENUM\\_DATA\\_TYPE](#). The *FieldType* parameter can be NULL if the type code is not required.

### Return Values

If the function succeeds, the return value is SAPI\_OK.

Some possible error codes are listed in the following table.

Value	Meaning
SAPI_ERR_SESSION_IS_NULL	SAPI session handle is NULL, was not created using <a href="#">SAPIHandleAcquire</a> , or is corrupt.
SAPI_ERR_NULL_VALUE_IS_NOT_ALLOWED	<i>ValueLen</i> or <i>CertHandle</i> is NULL and must not be NULL.

SAPI_ERR_CERT_HANDLE_NOT_INITIALIZED	The certificate handle is either not initialized or is corrupt.
SAPI_ERR_PARAMETER_LEN_IS_TOO_SHORT	The size of <i>Value</i> is too short to hold the required certificate field.
SAPI_ERR_UNSUPPORTED_CERT_FIELD	The required field is not supported by the function.
SAPI_ERR_CERT_FIELD_VALUE_NOT_FOUND	The value of the required field was not found in the certificate. This error code is usually returned for SAPI_ENUM_CERT_FIELD_EMAIL.
SAPI_ERR_FAILED_TO_GET_CERT_FIELD	A general error occurred while retrieving the certificate field. Call <a href="#">SAPIExtendedLastErrorGet</a> for more information about the exact problem.

**See Also**

[SAPIHandleAcquire](#), [SAPICertificatesEnumCont](#), [SAPICertificateGetDefault](#), [SAPICertificateGetFieldByBlob](#)

## SAPICertificateGetFieldByBlob

The **SAPICertificateGetFieldByBlob** function retrieves a single field from a certificate's raw data (ASN1 encoded).

```
SAPI int SAPICertificateGetFieldByBlob (  
    SAPI_SES_HANDLE          Handle,  
    unsigned char            *EncodedCertificate,  
    unsigned long            EncodedCertificateLen,  
    SAPI_ENUM_CERT_FIELD    FieldID,  
    unsigned char            *Value,  
    unsigned long            *ValueLen,  
    SAPI_ENUM_DATA_TYPE     *FieldType  
);
```

### Parameters

#### Handle

[in] Handle of the SAPI session created by using [SAPIHandleAcquire](#).

#### EncodedCertificate

[in] A buffer holding the ASN1 certificate.

#### EncodedCertificateLen

[in] The length in bytes of *EncodedCertificate*.

#### FieldID

[in] The ID of the certificate field to be retrieved. A description of all available IDs is contained in `SAPI_ENUM_CERT_FIELD`.

#### Value

[out] Pointer to a buffer holding the retrieved data according to the value of *FieldID*. This parameter can be NULL if only the value length is required.

#### ValueLen

[in/out] Pointer to a value that specifies the size allocated for the *Value* parameter. When the function returns, this value contains the size of the data copied to *Value*, or if *Value* is NULL, the size that needs to be allocated for *Value*.

#### FieldType

[out] Pointer to a value that receives a code indicating the type of data that was returned in *Value*. For a list of the possible type codes, see [SAPI\\_ENUM\\_DATA\\_TYPE](#). The *FieldType* parameter can be NULL if the type code is not required.

### Return Values

If the function succeeds, the return value is `SAPI_OK`.

Some possible error codes are listed in the following table.

Value	Meaning
<code>SAPI_ERR_SESSION_IS_NULL</code>	SAPI session handle is NULL, was not created using <a href="#">SAPIHandleAcquire</a> , or is corrupt.

Value	Meaning
SAPI_ERR_NULL_VALUE_IS_NOT_ALLOWED	<i>EncodedCertificate</i> or <i>ValueLen</i> is NULL and must not be NULL, and <i>EncodedCertificateLen</i> must be greater than 0.
SAPI_ERR_PARAMETER_LEN_IS_TOO_SHORT	The size of <i>Value</i> is too short to hold the required certificate field.
SAPI_ERR_UNSUPPORTED_CERT_FIELD	The required field is not supported by the function.
SAPI_ERR_CERT_FIELD_VALUE_NOT_FOUND	The value of the required field was not found in the certificate. This error code is usually returned for SAPI_ENUM_CERT_FIELD_EMAIL.
SAPI_ERR_FAILED_TO_GET_CERT_FIELD	A general error occurred while retrieving the certificate field. Call <a href="#">SAPIExtendedLastErrorGet</a> for more information about the exact problem.

**See Also**

[SAPIHandleAcquire](#), [SAPICertificateGetFieldByHandle](#)

## SAPIPKCS7BlobGetValue

The **SAPIPKCS7BlobGetValue** function retrieves a single field from a PKCS#7 formatted signature blob. One of the fields that can be retrieved is the signing certificate value. From the certificate value, the application can obtain some information about the signer. This can be useful after signing data (as opposed to a file), or before verifying signed data.

```
SAPI int SAPIPKCS7BlobGetValue(  
    SAPI_SES_HANDLE          Handle,  
    unsigned char            *PKCS7Blob,  
    unsigned long            PKCS7BlobLen,  
    SAPI_ENUM_PKCS7_FIELD   FieldID,  
    unsigned char            *Value,  
    unsigned long            *ValueLen,  
    SAPI_ENUM_DATA_TYPE     *FieldType  
);
```

### Parameters

#### *Handle*

[in] Handle of the SAPI session created by using [SAPIHandleAcquire](#).

#### *PKCS7Blob*

[in] A buffer holding the PKCS#7 formatted signature blob.

#### *PKCS7BlobLen*

[in] The length in bytes of *PKCS7Blob*.

#### *FieldID*

[in] The ID of the field to be retrieved. See [SAPI\\_ENUM\\_PKCS7\\_FIELD](#).

#### *Value*

[out] Pointer to a buffer holding the retrieved data according to the value of *FieldID*. This parameter can be NULL if only the value length is required.

#### *ValueLen*

[in/out] Pointer to a value that specifies the size allocated for the *Value* parameter. When the function returns, this value contains the size of the data copied to *Value*, or if *Value* is NULL, the size that needs to be allocated for *Value*.

#### *FieldType*

[out] Pointer to a value that receives a code indicating the type of data that was returned in *Value*. For a list of the possible type codes, see [SAPI\\_ENUM\\_DATA\\_TYPE](#). The *FieldType* parameter can be NULL if the type code is not required.

### Return Values

If the function succeeds, the return value is `SAPI_OK`.

Some possible error codes are listed in the following table.

Value	Meaning
SAPI_ERR_SESSION_IS_NULL	SAPI session handle is NULL, was not created using <a href="#">SAPIHandleAcquire</a> , or is corrupt.
SAPI_ERR_NULL_VALUE_IS_NOT_ALLOWED	<i>PKCS7Blob</i> or <i>ValueLen</i> is NULL and must not be NULL.
SAPI_ERR_PARAMETER_LEN_IS_TOO_SHORT	The size of <i>Value</i> is too short to hold the required certificate field.
SAPI_ERR_UNSUPPORTED_CERT_FIELD	The required field is not supported by the function.
SAPI_ERR_FAILED_TO_PARSE_PKCS7_BLOB	A general error occurred while retrieving the certificate from the signature data. Call <a href="#">SAPIExtendedLastErrorGet</a> for more information about the exact problem.

**See Also**

[SAPIHandleAcquire](#), [SAPICertificateGetFieldByBlob](#)

## SAPICertificateGUISelect

The **SAPICertificateGUISelect** function displays all the available certificates in a dialog box, enabling the user to view each certificate's details and to select one of the certificates. This function returns the certificate handle of the selected certificate.

```
SAPI int SAPICertificateGUISelect (
    SAPI_SES_HANDLE          Handle,
    unsigned long            Flags,
    SAPI_CERT_HANDLE        *CertHandle
);
```

### Parameters

#### Handle

[in] Handle of the SAPI session created by using [SAPIHandleAcquire](#).

#### Flags

[in] Flag values bit mask. For more information, refer to the following section in Appendix D: [Flags when Displaying Certificate List](#).

#### CertHandle

[out] Pointer to a handle of the selected certificate. This handle can be used when calling other certificate functions such as [SAPICertificateGetFieldByHandle](#) and [SAPICertificateSetDefault](#).

When *CertHandle* is not in use anymore, it is released by calling the [SAPIHandleRelease](#) function.

### Return Values

If the function succeeds, the return value is `SAPI_OK`.

Some possible error codes are listed in the following table.

Value	Meaning
<code>SAPI_ERR_SESSION_IS_NULL</code>	SAPI session handle is NULL, was not created using <a href="#">SAPIHandleAcquire</a> , or is corrupt.
<code>SAPI_ERR_NULL_VALUE_IS_NOT_ALLOWED</code>	<i>CertHandle</i> is NULL and must not be NULL.
<code>SAPI_ERR_NO_CERT_TO_SELECT_FROM</code>	No certificate was found. This might be due to incorrect credentials or to networking problems.
<code>SAPI_FAILED_TO_ALLOCATE_MEMORY</code>	CoSign Signature Local failed to allocate the necessary memory for the certificate handle. This might be due to a shortage in system resources.
<code>SAPI_ERR_GUI_SELECT_CERT_OPERATION_WAS_CANCELLED</code>	The user pressed the Cancel button or the Esc key in the Select Certificate dialog box.
<code>SAPI_ERR_GUI_SELECT_CERT_RETURNED_NO_CERT</code>	A general error occurred while looking for a certificate. Call <a href="#">SAPIExtendedLastErrorGet</a> for more information about the exact problem.

### Remarks

This function can be called only by applications that have the `InteractWithDesktop` property.

Some of the strings displayed in the Select Certificate dialog box (such as Title and Command) can be controlled by the application. To change these strings, use the [SAPIConfigurationValueSet](#) function with one or more of the following configuration values:

SAPI\_ENUM\_CONF\_ID\_SEL\_CERT\_TITLE, SAPI\_ENUM\_CONF\_ID\_SEL\_CERT\_CMD.

***See Also***

[SAPIHandleAcquire](#), [SAPICertificateGetFieldByHandle](#), [SAPIHandleRelease](#), [SAPICertificateSetDefault](#), [SAPIConfigurationValueSet](#)

## SAPICertificateSetDefault

The **SAPICertificateSetDefault** function sets the certificate pointed to by the certificate handle, as the default certificate. The default certificate is used for all signing operations in the current SAPI session handle. This function is used in the following scenarios:

To improve signing performance when multiple signing operations must be carried out with the same certificate in the same SAPI session.

In environments where there is more than one available certificate.

```
SAPI int SAPICertificateSetDefault (
    SAPI_SES_HANDLE      Handle,
    SAPI_CERT_HANDLE     CertHandle
);
```

### Parameters

#### Handle

[in] Handle of the SAPI session created by using [SAPIHandleAcquire](#).

#### CertHandle

[in] The default certificate handle created using [SAPICertificatesEnumCont](#).

If NULL, this function tries to automatically look for a certificate, and set it as default. See the [SAPICertificateGetDefault](#) function for more details about the algorithm for automatic certificate searching. If not NULL, *CertHandle* is released by calling the [SAPIHandleRelease](#) function, when it is not in use anymore.

### Return Values

If the function succeeds, the return value is SAPI\_OK.

Some possible error codes are listed in the following table.

Value	Meaning
SAPI_ERR_SESSION_IS_NULL	SAPI session handle is NULL, was not created using <a href="#">SAPIHandleAcquire</a> , or is corrupt.
SAPI_ERR_CERT_HANDLE_NOT_INITIALIZED	The certificate handle is either not initialized or is corrupt.
SAPI_ERR_NO_CERT_TO_SELECT_FROM	No certificate was found. This might be due to incorrect credentials, networking problems, or because the preferred certificate could not be found. <b>NOTE:</b> This error can occur only if <i>CertHandle</i> is NULL.
SAPI_ERR_TOO_MANY_CERTS_TO_SELECT_FROM	More than one certificate is available and CoSign Signature Local cannot choose between them. Use either <a href="#">SAPIConfigurationValueSet</a> or call the certificates enumeration functions and select the certificate with which CoSign Signature Local works. <b>NOTE:</b> This error can occur only if <i>CertHandle</i> is NULL.

Value	Meaning
SAPI_ERR_FAILED_TO_SET_CERT	A general error occurred while setting the default certificate. Call <a href="#">SAPIExtendedLastErrorGet</a> for more information about the exact problem.

**Remarks**

This function can be called more than once, each time with a different valid certificate handle.

To improve performance, CoSign Signature Local stores the certificate handle in the session's internal cache, as well as additional data that is related to this certificate, such as the certificate chain. Each time this function is called, all the stored data is erased. This means that calling this function twice with the same certificate handle might degrade performances because some of the cached data is recreated.

A certificate can also be set as default by using the [SAPICertificateGetDefault](#) function.

When setting a certificate as default, a copy of the certificate handle is created, therefore the application is still responsible for releasing the certificate handle that is passed to this function. The default certificate handle is automatically released when the session is closed.

**See Also**

[SAPIHandleAcquire](#), [SAPIHandleRelease](#), [SAPICertificateGetDefault](#), [SAPIConfigurationValueSet](#), [SAPICertificatesEnumCont](#)

## SAPICertificateGetDefault

The **SAPICertificateGetDefault** function employs a selection algorithm to choose a certificate, and returns a certificate handle of the chosen certificate.

The algorithm selects a certificate as follows: the algorithm goes through the following ordered set of criteria. When a criterion is met, the certificate meeting the criterion is the chosen certificate, and the process stops.

1. A certificate was explicitly set as default by [SAPICertificateSetDefault](#), or a certificate was automatically set as default after a signing operation, or even after calling this function.
2. A certificate with a subject key identifier equal to a subject key identifier that was set as a configuration value (SAPI\_ENUM\_CONF\_ID\_CERT\_SKI).
3. A certificate with a serial ID equal to a serial ID that was set as a configuration value (SAPI\_ENUM\_CONF\_ID\_CERT\_SERIAL\_ID).
4. There is only one available valid certificate.

This function also sets the chosen certificate as the default certificate for the current SAPI session, unless the SAPI\_ENUM\_CONF\_ID\_SET\_CERT\_AS\_DEFAULT configuration value is explicitly set to 0.

```
SAPI int SAPICertificateGetDefault (
    SAPI_SES_HANDLE      Handle,
    SAPI_CERT_HANDLE     *CertHandle
);
```

### Parameters

#### Handle

[in] Handle of the SAPI session created by using [SAPIHandleAcquire](#).

#### CertHandle

[out] Pointer to a handle of the chosen certificate. This handle can be used when calling other certificate functions such as [SAPICertificateGetFieldByHandle](#).

When *CertHandle* is not in use anymore, it is released by calling the [SAPIHandleRelease](#) function.

### Return Values

If the function succeeds, the return value is SAPI\_OK.

Some possible error codes are listed in the following table.

Value	Meaning
SAPI_ERR_SESSION_IS_NULL	SAPI session handle is NULL, was not created using <a href="#">SAPIHandleAcquire</a> , or is corrupt.
SAPI_ERR_NULL_VALUE_IS_NOT_ALLOWED	<i>CertHandle</i> is NULL and must not be NULL.
SAPI_ERR_NO_CERT_TO_SELECT_FROM	No certificate was found. This might be due to incorrect credentials, networking problems, or because the preferred certificate could not be found.
SAPI_ERR_TOO_MANY_CERTS_TO_SELECT_FROM	More than one certificate is available and CoSign Signature Local cannot choose between them. Use either <a href="#">SAPICertificateSetDefault</a> or

Value	Meaning
	<a href="#">SAPIConfigurationValueSet</a> to set the certificate with which CoSign Signature Local works.
SAPI_ERR_FAILED_TO_GET_CERT	A general error occurred while looking for a certificate. Call <a href="#">SAPIExtendedLastErrorGet</a> for more information about the exact problem.

### **Remarks**

To prevent the chosen certificate from becoming the default certificate, call the [SAPIConfigurationValueSet](#) function with the SAPI\_ENUM\_CONF\_ID\_SET\_CERT\_AS\_DEFAULT configuration value set to 0.

If either the SAPI\_ENUM\_CONF\_ID\_CERT\_SKI or the SAPI\_ENUM\_CONF\_ID\_CERT\_SERIAL\_ID configuration values are set but the respective certificate cannot be found, an error is returned. This is true even in cases where there is only one available certificate that could have been chosen if none of these values had been set.

### **See Also**

[SAPIHandleAcquire](#), [SAPICertificateGetFieldByHandle](#), [SAPIHandleRelease](#), [SAPIConfigurationValueSet](#), [SAPICertificateSetDefault](#)

## Graphical Image Objects

This chapter describes the functions used for graphical image objects. Graphical image objects are images that are centrally stored within DocuSign Signature Appliance for every user and are added to the visual representation of a signature field when it is signed. These functions fall into the following groups:

- Selecting and Retrieving Graphical Image Objects
- Managing and Affecting Graphical Image Objects

There are three types of graphical images:

- Graphical signature
- Initials

Logo. Starting from CoSign version 5.4, a single logo is supported.

All graphical images that were generated prior to CoSign version 4.6 are marked as Graphical signatures.

Starting from CoSign version 5.4, a new standard signing ceremony dialog enables selecting and entering data to be used in the digital signature act. This dialog also enables managing all the user's graphical signatures. From version 5.4, the source of a graphical signature can be either the CoSign appliance or specified local directories of the user. The option of using local directories overcomes the CoSign appliance's strict limitations on image size and on the number of graphical signatures per user.

### Selecting and Retrieving Graphical Image Objects

The following functions enable the application to select a graphical image in a multi graphical image object environment and retrieve a selected graphical object.

You can use any one of the following methods for selecting a graphical image:

Request the default image by calling [SAPIGraphicSigImageGetDefaultEx](#). This method works if:

- ◆ The user has only one graphical image in DocuSign Signature Appliance.
- ◆ A default graphical image was set earlier in this session by calling either [SAPIGraphicSigImageSetDefaultEx](#), [SAPIGraphicSigImageGUISelect](#), or [SAPISigningCeremonyGUI](#).

Let the user view all the graphical images and select the desired image by calling [SAPIGraphicSigImageGUISelect](#).

Enumerate all the graphical objects, retrieve each image's information, analyze the information and choose the one that best fits the user's needs. The image enumeration is done by first calling [SAPIGraphicSigImageEnumInit](#) and then making consecutive calls to [SAPIGraphicSigImageEnumCont](#). Retrieval of image information for analysis is done by calling [SAPIGraphicSigImageInfoGet](#).

After the desired image is selected, it can be set as the default, so further calls to the API such as [SAPISignatureFieldSignEx](#) use this image, or it can be retrieved and placed elsewhere, depending on the requirements of the calling application.

The following functions are included in this group:

[SAPIGraphicSigImageEnumInit](#), [SAPIGraphicSigImageEnumCont](#), [SAPIGraphicSigImageGUISelect](#), [SAPIGraphicSigImageInfoGet](#), [SAPIGraphicSigImageGetDefault](#), and [SAPIGraphicSigImageSetDefault](#)

**Note:** If a user has only one graphical image and this image is used by [SAPISignatureFieldSign](#), there is no need to call any of the above functions, since the [SAPISignatureFieldSign](#) function automatically selects the only image that exists.

## Managing Graphical Image Objects

The following functions enable the application to create, update, and delete a specific user's graphical image objects.

To create or update a Graphical signature's image, fill in the fields of the graphical image structure and call [SAPIGraphicSigImageInfoCreate](#) and [SAPIGraphicSigImageInfoUpdate](#) respectively.

To create or update an Initials graphical image, fill in the fields of the graphical image structure and call [SAPIGraphicSigImageInfoCreate](#) and [SAPIGraphicSigImageInfoUpdate](#) respectively.

To create or update a Logo graphical image, fill in the fields of the graphical image structure and call [SAPIGraphicSigImageInfoCreate](#) and [SAPIGraphicSigImageInfoUpdate](#) respectively.

To delete an image, the application selects the desired image for deletion (see [Selecting and Retrieving Graphical Image Objects](#)) and then uses the image handle as an input for [SAPIGraphicSigImageInfoDelete](#).

**Note:** A user can create, update, and delete one's own graphical images using the Graphical Signatures application found in the Control Panel. In most cases, it is not necessary to write a proprietary application to manage the user's graphical images. For more information, See *Using the Graphical Signature Management Application* in *Chapter 4: Deploying the Client* in the *DocuSign Signature Appliance Administrator Guide*.

## SAPIGraphicSigImageEnumInit

The **SAPIGraphicSigImageEnumInit** function initializes the graphical images enumeration operation. This operation is used in environments where there may be more than one available image. A call to the **SAPIGraphicSigImageEnumInit** function is followed by calls to [SAPIGraphicSigImageEnumCont](#) function until all the images are retrieved, or until the required graphical image object is found. To retrieve more information about a specific graphical image, call the [SAPIGraphicSigImageInfoGet](#) function.

```
SAPI int SAPIGraphicSigImageEnumInit (
    SAPI_SES_HANDLE      Handle,
    SAPI_CONTEXT         *GraphicImageContext,
    unsigned long        FormatsPermitted,
    unsigned long        Flags
);
```

### Parameters

#### *Handle*

[in] Handle of the SAPI session created using [SAPIHandleAcquire](#).

#### *GraphicImageContext*

[out] Pointer to the graphical images enumeration operation context. This context contains internal information about the operation. The information in this context is passed in all subsequent calls to [SAPIGraphicSigImageEnumCont](#). Changing the value of *GraphicImageContext* by the application is not allowed since it can lead to unexpected behavior.

*GraphicImageContext* must be released by the

SAPIContextRelease function when the graphical images enumeration operation is complete.

### **FormatsPermitted**

[in] Bit flag parameter that indicates the formats of the graphical objects that are retrieved by the enumeration operation. You may set one or more bits. For the possible settings for *FormatsPermitted*, see [Graphical Signature Image Related Flags](#).

### **Flags**

[in] Flag values bit mask. For more information, refer to the following section in Appendix D: [Flags when Enumerating or Using Graphical Signatures](#).

### **Return Values**

If the function succeeds, the return value is SAPI\_OK.

Some possible error codes are listed in the following table.

Value	Meaning
SAPI_ERR_SESSION_IS_NULL	SAPI session handle is NULL, was not created using <a href="#">SAPIHandleAcquire</a> , or is corrupt.
SAPI_ERR_NULL_VALUE_IS_NOT_ALLOWED	<i>GraphicImageContext</i> is NULL and must not be NULL.
SAPI_FAILED_TO_ALLOCATE_MEMORY	CoSign Signature Local failed to allocate the necessary memory for the new operation context. This might be due to a shortage in system resources.

### **Remarks**

**SAPIGraphicSigImageEnumInit** builds an internal list of graphical image objects according to the available objects at the moment it is called. Only objects from the list are returned by the [SAPIGraphicSigImageEnumCont](#) function. Therefore, new graphical image objects that become available after the **SAPIGraphicSigImageEnumInit** returns do not affect the graphical image object handles that are retrieved by the [SAPIGraphicSigImageEnumCont](#) function. However, removing objects after **SAPIGraphicSigImageEnumInit** returns can cause [SAPIGraphicSigImageEnumCont](#) to fail.

In environments where there is only one graphical image object, the graphical image objects enumeration operation is unnecessary.

### **See Also**

[SAPIHandleAcquire](#),

SAPIContextRelease, [SAPIGraphicSigImageEnumCont](#), [SAPIGraphicSigImageGUISelect](#)

## SAPIGraphicSigImageEnumCont

The **SAPIGraphicSigImageEnumCont** function continues the graphical image objects enumeration operation. It returns a pointer to a single graphical image object handle from the internal objects list built by the [SAPIGraphicSigImageEnumInit](#) function. Every call returns a new object handle until the end of the list is reached.

```
SAPI int SAPIGraphicSigImageEnumCont (
    SAPI_SES_HANDLE      Handle,
    SAPI_CONTEXT         *GraphicImageContext,
    SAPI_GR_IMG_HANDLE   *GraphicImageHandle
);
```

### **Parameters**

#### *Handle*

[in] Handle of the SAPI session created using [SAPIHandleAcquire](#).

#### *GraphicImageContext*

[in] Pointer to the graphical images enumeration operation context created using [SAPIGraphicSigImageEnumInit](#).

*GraphicImageContext* must be released by the

SAPIContextRelease function when there are no more objects to retrieve or when the required object was found.

### *GraphicImageHandle*

[out] Pointer to a handle of a graphical image object. This handle can be used when calling other graphic image functions, such as [SAPIGraphicSigImageInfoGet](#), [SAPIGraphicSigImageInfoDelete](#), and [SAPIGraphicSigImageSetDefault](#).

When *GraphicImageHandle* is no longer in use, it is released by calling the [SAPIHandleRelease](#) function.

### **Return Values**

If the function succeeds, the return value is SAPI\_OK.

Some possible error codes are listed in the following table.

Value	Meaning
SAPI_ERR_SESSION_IS_NULL	SAPI session handle is NULL, was not created using <a href="#">SAPIHandleAcquire</a> , or is corrupt.
SAPI_ERR_NULL_VALUE_IS_NOT_ALLOWED	<i>GraphicImageContext</i> or <i>GraphicImageHandle</i> is NULL and must not be NULL.
SAPI_FAILED_TO_ALLOCATE_MEMORY	CoSign Signature Local failed to allocate the necessary memory for the graphical image object handle. This might be due to a shortage in system resources.
SAPI_ERR_CONTEXT_NOT_INITIALIZED	The graphical image object enumeration context is either not initialized or is corrupt.
SAPI_ERROR_NO_MORE_ITEMS	There are no more objects to retrieve.

### **Remarks**

**SAPIGraphicSigImageEnumCont** returns only the objects that were available when [SAPIGraphicSigImageEnumInit](#) was called.

In order to retrieve the whole objects list, **SAPIGraphicSigImageEnumCont** is called until SAPI\_ERROR\_NO\_MORE\_ITEMS is returned.

### **See Also**

[SAPIHandleAcquire](#),

SAPIContextRelease, [SAPIGraphicSigImageEnumInit](#), [SAPIGraphicSigImageGUISelect](#),  
[SAPIGraphicSigImageSetDefault](#), [SAPIGraphicSigImageInfoDelete](#), [SAPIGraphicSigImageInfoGet](#)

## SAPIGraphicSigImageGUISelect

The **SAPIGraphicSigImageGUISelect** function displays all the available graphical image objects for the user in a dialog box, enabling the user to view and select an image with which to sign. This dialog box can also be used for adding new graphical image objects from a file or from a special pad, and for deleting and updating existing objects.

```
SAPI int SAPIGraphicSigImageGUISelect (
    SAPI_SES_HANDLE          Handle,
    unsigned long           FormatsPermitted,
    unsigned long           Flags,
    SAPI_ENUM_GR_IMG_SELECT_MODE Mode,
    SAPI_GR_IMG_HANDLE      *GraphicImageHandle
);
```

### Parameters

#### Handle

[in] Handle of the SAPI session created using [SAPIHandleAcquire](#).

#### FormatsPermitted

[in] Bit flag parameter that indicates the formats of displayed graphical objects. You may set one or more bits. For the possible settings for *FormatsPermitted*, see [Graphical Signature Image Related Flags](#).

#### Flags

[in] Reserved for future use and must be zero.

#### Mode

[in] This parameter indicates in which mode the graphic image selection dialog box opens. There are three different modes, as described in [SAPI\\_ENUM\\_GR\\_IMG\\_SELECT\\_MODE](#):

**Selection as part of the signature ceremony** – This mode enables the user to select a graphical image that will be included in the signature. If no image is available, a temporary graphical image will be displayed. However, if the user does not have a certificate, or has multiple certificates, not even a temporary image is displayed.

To work in this mode, the value of this parameter is `SAPI_ENUM_GR_IMG_SEL_MODE_SELECT`.

**User mode** – This mode enables the user to manage and edit graphical images. The dialog box opens, displaying all the available images. The user is then able to add, delete, or update graphical images. In this mode, the user cannot select an image.

To work in this mode, the value of this parameter is `SAPI_ENUM_GR_IMG_SEL_MODE_VIEW_USER`.

**Administrative mode (or Kiosk mode)** – This mode is similar to User mode regarding operations that can be performed on graphical images. The only difference is that in this mode a user is required to supply credentials before accessing graphical image objects. This mode is useful when a single machine serves multiple users, for example a machine with a signature pad attached that is located in the Human Resources department and is accessed by users in other departments to capture their graphical signature and load it to DocuSign Signature Appliance.

To work in this mode, the value of this parameter is `SAPI_ENUM_GR_IMG_SEL_MODE_VIEW_KIOSK`.

**Volatile mode** – In this mode, a graphical image is captured for the purpose of a digital or electronic signature operation. This image is not kept in the DocuSign Signature Appliance or on user’s disk.

### *GraphicImageHandle*

[out] Pointer to a handle of a graphical image object.

This parameter is available only when the value of *Mode* is SAPI\_ENUM\_GR\_IMG\_SEL\_MODE\_SELECT. In all other modes, this parameter is ignored and is NULL.

This handle can later be used when calling other graphic image functions, such as [SAPIGraphicSigImageInfoGet](#), [SAPIGraphicSigImageInfoDelete](#), and [SAPIGraphicSigImageSetDefault](#).

When *GraphicImageHandle* is no longer in use, it is released by calling [SAPIHandleRelease](#).

### **Return Values**

If the function succeeds, the return value is SAPI\_OK.

Some possible error codes are listed in the following table.

Value	Meaning
SAPI_ERR_SESSION_IS_NULL	SAPI session handle is NULL, was not created using <a href="#">SAPIHandleAcquire</a> , or is corrupt.
SAPI_ERR_NULL_VALUE_IS_NOT_ALLOWED	<i>GraphicImageHandle</i> is NULL and must not be NULL when in <i>Selection mode</i> .
SAPI_ERR_TOKEN_NOT_SUPPORTED	Either the session handle is corrupt or the user is logged in to a token that does not support this functionality. Only DocuSign Signature Appliance supports this functionality.
SAPI_ERR_OPERATION_CANCELLED_BY_USER	No graphical image was selected. The user pressed the Cancel button before making a selection.
SAPI_ERR_NO_ITEMS	There are no objects to select.

### **Remarks**

This function can be called only by applications that have the *InteractWithDesktop* property.

If the function was called in Selection mode and the return code is SAPI\_OK, the selected image is also set as the default.

Starting from CoSign version 5.4, it is recommended to use the [SAPISigningCeremonyGUI](#) function.

### **See Also**

[SAPIHandleAcquire](#), [SAPIGraphicSigImageEnumInit](#), [SAPIGraphicSigImageEnumCont](#), [SAPIGraphicSigImageSetDefault](#), [SAPIGraphicSigImageInfoGet](#), [SAPIGraphicSigImageInfoDelete](#)

## SAPIGraphicSigImageSetDefaultEx

The **SAPIGraphicSigImageSetDefaultEx** function sets the graphical image associated with the graphic image handle as the default graphic signature image. The default graphic signature image is used for all signing operations in the current SAPI session handle. This function is used in cases where the user might have more than one available graphical image object.

```
SAPI int SAPIGraphicSigImageSetDefaultEx (  
    SAPI_SES_HANDLE          Handle,  
    SAPI_GR_IMG_HANDLE      GraphicImageHandle,  
    unsigned long           Flags  
);
```

### **Parameters**

#### *Handle*

[in] Handle of the SAPI session created using [SAPIHandleAcquire](#).

#### *GraphicImageHandle*

[in] The default certificate handle created using one of the following functions:

[SAPIGraphicSigImageEnumCont](#), [SAPIGraphicSigImageGUISelect](#), or [SAPIGraphicSigImageInfoCreate](#).

If this parameter is NULL, the function automatically looks for a graphical image object and sets it as the default. See Selection mode in the [SAPIGraphicSigImageGUISelect](#) function for more details about the automatic search for a default image.

#### *Flags*

[in] The type of graphical image (Graphical, Initials, or Logo). For the exact value, see [Flags when Enumerating or Using Graphical Signatures](#).

### **Return Values**

If the function succeeds, the return value is SAPI\_OK.

Some possible error codes are listed in the following table.

Value	Meaning
SAPI_ERR_SESSION_IS_NULL	SAPI session handle is NULL, was not created using <a href="#">SAPIHandleAcquire</a> , or is corrupt.
SAPI_ERR_TOO_MANY_ITEMS	More than one graphical image is available and CoSign Signature Local cannot choose between them. Use either <a href="#">SAPIGraphicSigImageGUISelect</a> or enumerate the images with <a href="#">SAPIGraphicSigImageEnumInit</a> and use <a href="#">SAPIGraphicSigImageEnumCont</a> to select the image that is set as the default. <b>NOTE:</b> This error can occur only if <i>GraphicImageHandle</i> is NULL.
SAPI_ERR_NO_ITEMS	No image was found. This might be due to incorrect credentials, networking problems, or because the user has no graphical images stored. <b>NOTE:</b> This error can occur only if <i>GraphicImageHandle</i> is NULL.
SAPI_ERR_FAILED_TO_ALLOCATE_MEMORY	CoSign Signature Local failed to allocate the necessary memory for the default image. This might be due to a shortage in system resources.
SAPI_ERR_INVALID_HANDLE	<i>GraphicImageHandle</i> is corrupt or not initialized correctly.

### **Remarks**

This function can be called more than once, each time with a different valid graphical image handle.

### **See Also**

[SAPIHandleAcquire](#), [SAPIGraphicSigImageEnumInit](#), [SAPIGraphicSigImageEnumCont](#), [SAPIGraphicSigImageGetDefault](#), [SAPIGraphicSigImageGUISelect](#), [SAPIGraphicSigImageSetDefaultEx](#)

## SAPIGraphicSigImageSetDefault

The **SAPIGraphicSigImageSetDefault** function is the older version of the [SAPIGraphicSigImageSetDefaultEx](#) function.

```
SAPI int SAPIGraphicSigImageSetDefault (
    SAPI_SES_HANDLE          Handle,
    SAPI_GR_IMG_HANDLE      *GraphicImageHandle
);
```

## SAPIGraphicSigImageGetDefaultEx

The **SAPIGraphicSigImageGetDefaultEx** employs a selection algorithm to choose a graphical signature image, and returns the handle of the chosen image. The algorithm selects a graphical signature image by interrogating an ordered set of criteria. When a criterion is met, the graphical image meeting the criterion is the chosen image, and the process stops. The ordered criteria are:

1. A graphical image was previously set as default in one of the following ways:
  - ◆ A graphical signature image was explicitly set as the default by [SAPIGraphicSigImageSetDefault](#).
  - ◆ A graphical image was selected by the user when [SAPIGraphicSigImageGUISelect](#) was called.
  - ◆ A graphical image was set because of a previous call to [SAPIGraphicSigImageGetDefault](#).
2. A graphical image with a name equal to a name that was set as a configuration value (SAPI\_ENUM\_CONF\_ID\_GR\_SIG\_PREF\_NAME) is found.
3. There is only one available graphical image.

This function also sets the selected image as the default image for the current SAPI session.

```
SAPI int SAPIGraphicSigImageGetDefaultEx (
    SAPI_SES_HANDLE          Handle,
    SAPI_GR_IMG_HANDLE      *GraphicImageHandle,
    Unsigned Long           Flags
);
```

### Parameters

#### Handle

[in] Handle of the SAPI session created using [SAPIHandleAcquire](#).

#### GraphicImageHandle

[out] Pointer to a handle of the selected graphical signature image. This handle can be used when calling other graphical image functions such as [SAPIGraphicSigImageInfoUpdate](#) and [SAPIGraphicSigImageInfoDelete](#).

When *GraphicImageHandle* is not in use anymore, it is released by calling the [SAPIHandleRelease](#) function.

#### Flags

[in] The type of graphical image (Graphical, Initials, or Logo). For the exact value, see [Flags when Enumerating or Using Graphical Signatures](#)

### Return Values

If the function succeeds, the return value is SAPI\_OK.

Some possible error codes are listed in the following table.

Value	Meaning
SAPI_ERR_SESSION_IS_NULL	SAPI session handle is NULL, was not created using <a href="#">SAPIHandleAcquire</a> , or is corrupt.

Value	Meaning
SAPI_ERR_NULL_VALUE_IS_NOT_ALLOWED	<i>GraphicImageHandle</i> is NULL and must not be NULL.
SAPI_ERR_NO_ITEMS	No graphical image was found. This might be due to incorrect credentials, networking problems, or because the preferred image name could not be found.
SAPI_FAILED_TO_ALLOCATE_MEMORY	CoSign Signature Local failed to allocate the necessary memory for the graphical image object handle. This might be due to a shortage in system resources.

### **Remarks**

If SAPI\_ENUM\_CONF\_ID\_GR\_SIG\_PREF\_NAME is set but the respective graphical signature image cannot be found, an error is returned even in cases where there is only one available image that could have been selected if this value had not been set.

### **See Also**

[SAPIHandleAcquire](#), [SAPIGraphicSigImageInfoUpdate](#), [SAPIGraphicSigImageInfoDelete](#), [SAPIGraphicSigImageGUISelect](#), [SAPIHandleRelease](#), [SAPIConfigurationValueSet](#)

## SAPIGraphicSigImageGetDefault

The **SAPIGraphicSigImageGetDefault** function is the older version of the [SAPIGraphicSigImageGetDefaultEx](#) function.

```
SAPI int SAPIGraphicSigImageGetDefault (
    SAPI_SES_HANDLE          Handle,
    SAPI_GR_IMG_HANDLE       *GraphicImageHandle
);
```

## SAPIGraphicSigImageInfoGet

The **SAPIGraphicSigImageInfoGet** function retrieves all the data of a graphical signature image associated with the provided graphical image handle.

```
SAPI int SAPIGraphicSigImageInfoGet (
    SAPI_SES_HANDLE          Handle,
    SAPI_GR_IMG_HANDLE      GraphicImageHandle,
    SAPI_GR_IMG_INFO        *GraphicImageInfo,
    SAPI_ENUM_GRAPHIC_IMAGE_FORMAT Convert,
    unsigned long           Flags
);
```

### Parameters

#### Handle

[in] Handle of the SAPI session created using [SAPIHandleAcquire](#).

#### GraphicImageHandle

[in] A graphical image handle. The function returns the information of the graphical image associated with this handle. This handle is created using one of the following functions: [SAPIGraphicSigImageEnumCont](#), [SAPIGraphicSigImageGUISelect](#), [SAPIGraphicSigImageInfoCreate](#), or [SAPIGraphicSigImageGetDefault](#).

#### GraphicImageInfo

[out] A pointer to a structure that contains the graphical image's information.

Before calling the function, the *StructLen* member of the structure must be set to the size of the structure.

#### Convert

[in] This parameter indicates whether the graphical image needs to be converted before retrieval.

The conversion includes converting from certain BMP types to others, or converting from raster format to vector format. This conversion does not transform the BMP format to JPEG format, and vice versa. The *ImageConvertedFormat* member of *GraphicImageInfo* is set with the value of the *Convert* parameter.

Typically, this parameter is set to `SAPI_ENUM_GRAPHIC_IMAGE_NONE`, indicating that the image is to be returned in the same format in which it is stored.

#### Flags

[in] Flag values bit mask. The available values are:

**AR\_GR\_SIG\_GET\_INTERNAL\_ALLOC**: This flag indicates that it is the responsibility of the function to allocate memory for the *GraphicImage* member in *GraphicImageInfo*.

If this flag is not set and the *GraphicImage* member is `NULL`, the function returns the number of bytes allocated to hold the graphical signature image in the *GraphicImageLen* member of *GraphicImageInfo*.

If this flag is not set and the *GraphicImage* member of *GraphicImageInfo* is not `NULL`, the function fails if the size indicated by the *GraphicImageLen* member of *GraphicImageInfo* is smaller than the required length.

### Return Values

If the function succeeds, the return value is `SAPI_OK`.

Some possible error codes are listed in the following table.

Value	Meaning
SAPI_ERR_SESSION_IS_NULL	SAPI session handle is NULL, was not created using <a href="#">SAPIHandleAcquire</a> , or is corrupt.
SAPI_ERR_NULL_VALUE_IS_NOT_ALLOWED	<i>GraphicImageHandle</i> or <i>GraphicImageInfo</i> is NULL and must not be NULL.
SAPI_ERR_INVALID_HANDLE	The <i>GraphicImageHandle</i> is corrupt or not initialized correctly.
SAPI_FAILED_TO_ALLOCATE_MEMORY	CoSign Signature Local failed to allocate the necessary memory for the graphical image information structure. This might be due to a shortage in system resources.  This error can occur only if AR_GR_SIG_GET_INTERNAL_ALLOC is set in <i>Flags</i> .

### Remarks

When *GraphicImageInfo* is no longer required, release any dynamically allocated memory. If the allocation was done by the **SAPIGraphicSigImageInfoGet** function, release it by calling [SAPIStructRelease](#).

**Note:** [SAPIStructRelease](#) must not be called if the allocation was not done by the **SAPIGraphicSigImageInfoGet** function.

### See Also

[SAPIHandleAcquire](#), [SAPIGraphicSigImageGUISelect](#), [SAPIHandleRelease](#), [SAPIGraphicSigImageEnumCont](#), [SAPIGraphicSigImageInfoCreate](#), [SAPIGraphicSigImageGetDefault](#), [SAPIStructRelease](#)

## SAPIGraphicSigImageInfoCreate

The **SAPIGraphicSigImageInfoCreate** function creates a new graphical signature image in DocuSign Signature Appliance. This new image becomes available for all applications that require a graphical signature image from DocuSign Signature Appliance.

The newly created image will be of type Graphical signature.

```
SAPI int SAPIGraphicSigImageInfoCreate (
    SAPI_SES_HANDLE          Handle,
    SAPI_GR_IMG_HANDLE      *GraphicImageHandle,
    SAPI_GR_IMG_INFO        *GraphicImageInfo
);
```

### Parameters

#### Handle

[in] Handle of the SAPI session created using [SAPIHandleAcquire](#).

#### GraphicImageHandle

[out] A pointer to a graphical signature image handle of the image that was created.

#### GraphicImageInfo

[in/out] A pointer to a structure that contains the new graphical image information.

The mandatory structure members for this function are: *GraphicImage*, *GraphicImageLen*, *szGraphicImageName*, and *StructLen* (see [SAPI GR IMG INFO](#)). All other members are ignored as input parameters.

*Width*, *Height*, and *Format* structure members might be changed by the function based on the image format. The *Data Type* field in the *GraphicImageInfo* structure must be set to the value of `AR_GR_FLAG_IMAGE_TYPE`.

### Return Values

If the function succeeds, the return value is `SAPI_OK`.

Some possible error codes are listed in the following table.

Value	Meaning
<code>SAPI_ERR_SESSION_IS_NULL</code>	SAPI session handle is NULL, was not created using <a href="#">SAPIHandleAcquire</a> , or is corrupt.
<code>SAPI_ERR_NULL_VALUE_IS_NOT_ALLOWED</code>	<i>GraphicImageHandle</i> or <i>GraphicImageInfo</i> is NULL and must not be NULL.

### Remarks

The name of a graphical image cannot be NULL or based on white characters.

### See Also

[SAPIHandleAcquire](#), [SAPIHandleRelease](#), [SAPIGraphicSigImageGetDefault](#), [SAPIGraphicSigLogoInfoCreate](#), [SAPIGraphicSigInitialsInfoCreate](#)

## SAPIGraphicSigLogoInfoCreate

The **SAPIGraphicSigLogoInfoCreate** function is very similar to the **SAPIGraphicSigImageInfoCreate** function. In this case, the new graphical image is of type Logo.

```
SAPI int SAPIGraphicSigLogoInfoCreate (
    SAPI_SES_HANDLE           Handle,
    SAPI_GR_IMG_HANDLE       *GraphicImageHandle,
    SAPI_GR_IMG_INFO         *GraphicImageInfo
);
```

### Parameters

#### Handle

[in] Handle of the SAPI session created using [SAPIHandleAcquire](#).

#### GraphicImageHandle

[out] A pointer to a graphical signature image handle of the image that was created.

#### GraphicImageInfo

[in/out] A pointer to a structure that contains the new graphical image information.

The mandatory structure members for this function are: *GraphicImage*, *GraphiImageLen*, *szGraphicImageName*, and *StructLen* (see [SAPI\\_GR\\_IMG\\_INFO](#)). All other members are ignored as input parameters.

*Width*, *Height*, and *Format* structure members might be changed by the function based on the image format.

### Return Values

If the function succeeds, the return value is `SAPI_OK`.

Some possible error codes are listed in the following table.

Value	Meaning
<code>SAPI_ERR_SESSION_IS_NULL</code>	SAPI session handle is NULL, was not created using <a href="#">SAPIHandleAcquire</a> , or is corrupt.
<code>SAPI_ERR_NULL_VALUE_IS_NOT_ALLOWED</code>	<i>GraphicImageHandle</i> or <i>GraphicImageInfo</i> is NULL and must not be NULL.

### Remarks

The name of a graphical image cannot be NULL or based on white characters.

### See Also

[SAPIHandleAcquire](#), [SAPIHandleRelease](#), [SAPIGraphicSigImageGetDefault](#), [SAPIGraphicSigImageInfoCreate](#), [SAPIGraphicSigInitialsInfoCreate](#)

## SAPIGraphicSigInitialsInfoCreate

The **SAPIGraphicSigInitialsInfoCreate** function is very similar to the [SAPIGraphicSigImageInfoCreate](#) function. In this case, the new graphical image is of type Initials.

```
SAPI int SAPIGraphicSigInitialsInfoCreate (
    SAPI_SES_HANDLE          Handle,
    SAPI_GR_IMG_HANDLE      *GraphicImageHandle,
    SAPI_GR_IMG_INFO        *GraphicImageInfo
);
```

### Parameters

#### Handle

[in] Handle of the SAPI session created using [SAPIHandleAcquire](#).

#### GraphicImageHandle

[out] A pointer to a graphical signature image handle of the image that was created.

#### GraphicImageInfo

[in/out] A pointer to a structure that contains the new graphical image information.

The mandatory structure members for this function are: *GraphicImage*, *GraphicImageLen*, *szGraphicImageName*, and *StructLen* (see [SAPI\\_GR\\_IMG\\_INFO](#)). All other members are ignored as input parameters.

*Width*, *Height*, and *Format* structure members might be changed by the function based on the image format.

### Return Values

If the function succeeds, the return value is `SAPI_OK`.

Some possible error codes are listed in the following table.

Value	Meaning
<code>SAPI_ERR_SESSION_IS_NULL</code>	SAPI session handle is NULL, was not created using <a href="#">SAPIHandleAcquire</a> , or is corrupt.
<code>SAPI_ERR_NULL_VALUE_IS_NOT_ALLOWED</code>	<i>GraphicImageHandle</i> or <i>GraphicImageInfo</i> is NULL and must not be NULL.

### Remarks

The name of a graphical image cannot be NULL or based on white characters.

### See Also

[SAPIHandleAcquire](#), [SAPIHandleRelease](#), [SAPIGraphicSigImageGetDefault](#), [SAPIGraphicSigImageInfoCreate](#), [SAPIGraphicSigLogoInfoCreate](#)

## SAPIGraphicSigImageInfoDelete

The **SAPIGraphicSigImageInfoDelete** function deletes the graphical signature image that is associated with the provided graphical image handle from the DocuSign Signature Appliance. After the call to this function, the graphical image that was deleted is no longer available for use, whether you are in the current SAPI session or any other application.

```
SAPI int SAPIGraphicSigImageInfoDelete (
    SAPI_SES_HANDLE          Handle,
    SAPI_GR_IMG_HANDLE      GraphicImageHandle
);
```

### Parameters

#### Handle

[in] Handle of the SAPI session created using [SAPIHandleAcquire](#).

#### GraphicImageHandle

[in] Graphical signature image handle that needs to be deleted. This handle was created using one of the following functions: [SAPIGraphicSigImageEnumCont](#), [SAPIGraphicSigImageGUISelect](#), [SAPICertificatesEnumCont](#), or [SAPIGraphicSigImageGetDefault](#).

### Return Values

If the function succeeds, the return value is `SAPI_OK`.

Some possible error codes are listed in the following table.

Value	Meaning
<code>SAPI_ERR_SESSION_IS_NULL</code>	SAPI session handle is NULL, was not created using <a href="#">SAPIHandleAcquire</a> , or is corrupt.
<code>SAPI_ERR_NULL_VALUE_IS_NOT_ALLOWED</code>	<i>GraphicImageHandle</i> is NULL and must not be NULL.
<code>SAPI_ERR_INVALID_HANDLE</code>	The <i>GraphicImageHandle</i> is corrupt or not initialized correctly.

### Remarks

After calling this function, the graphical signature image might remain temporarily available for some applications because of the caching mechanism. It is not recommended to use a graphical image after its deletion because the results might be unpredictable.

The *GraphicImageHandle* is released using [SAPIHandleRelease](#) after the image is deleted.

### See Also

[SAPIHandleAcquire](#), [SAPIGraphicSigImageGUISelect](#), [SAPIHandleRelease](#), [SAPIGraphicSigImageEnumCont](#), [SAPICertificatesEnumCont](#), [SAPIGraphicSigImageGetDefault](#)

## SAPIGraphicSigImageInfoUpdate

The **SAPIGraphicSigImageInfoUpdate** function changes the data of a graphical signature image, its name, or both the data and name. This function updates the image associated with the provided graphical image handle.

The updated image should be of type Graphical signature.

```
SAPI int SAPIGraphicSigImageInfoUpdate (
    SAPI_SES_HANDLE           Handle,
    SAPI_GR_IMG_HANDLE       GraphicImageHandle,
    SAPI_GR_IMG_INFO         *GraphicImageInfo,
    unsigned long             Flags
);
```

### Parameters

#### Handle

[in] Handle of the SAPI session created using [SAPIHandleAcquire](#).

#### GraphicImageHandle

[in] A graphical image handle of the graphical signature image that needs to be updated. This handle was created using one of the following functions: [SAPIGraphicSigImageEnumCont](#), [SAPIGraphicSigImageGUISelect](#), [SAPICertificatesEnumCont](#), or [SAPIGraphicSigImageGetDefault](#).

#### GraphicImageInfo

[in] A pointer to a structure that contains the new graphical image information.

The mandatory structure members for this function are: *GraphicImage*, *GraphiImageLen*, *szGraphicImageName*, and *StructLen* (see [SAPI GR\\_IMG\\_INFO](#)). All other members are ignored.

#### Flags

[in] Reserved for future use and must be set to zero.

### Return Values

If the function succeeds, the return value is `SAPI_OK`.

Some possible error codes are listed in the following table.

Value	Meaning
<code>SAPI_ERR_SESSION_IS_NULL</code>	SAPI session handle is NULL, was not created using <a href="#">SAPIHandleAcquire</a> , or is corrupt.
<code>SAPI_ERR_NULL_VALUE_IS_NOT_ALLOWED</code>	<i>GraphicImageHandle</i> or <i>GraphicImageInfo</i> is NULL and must not be NULL.
<code>SAPI_ERR_INVALID_HANDLE</code>	The <i>GraphicImageHandle</i> is corrupt or not initialized correctly,

### Remarks

The name of a graphical signature image may not be NULL and cannot be composed of white characters only.

The only exceptions are graphical signature images created in a CoSign version prior to CoSign version 4.0. Such images were created with an empty name and therefore can be updated with the same value.

When updating a graphical signature image, verify that the image format is not damaged. Damage to the image format will prevent CoSign Signature Local from retrieving information such as width and height when [SAPIGraphicSigImageInfoGet](#) is subsequently called, and may cause unexpected results.

***See Also***

[SAPIHandleAcquire](#), [SAPIGraphicSigImageGUISelect](#), [SAPIHandleRelease](#),  
[SAPIGraphicSigImageEnumCont](#), [SAPICertificatesEnumCont](#), [SAPIGraphicSigImageGetDefaultEx](#),  
[SAPIGraphicSigInitialsInfoUpdate](#), [SAPIGraphicSigLogoInfoUpdate](#)

## SAPIGraphicSigLogoInfoUpdate

The **SAPIGraphicSigLogoInfoUpdate** function changes the data of a Logo graphical signature image, its name, or both the data and name. This function updates the image associated with the provided graphical image handle.

```
SAPI int SAPIGraphicSigLogoInfoUpdate (
    SAPI_SES_HANDLE           Handle,
    SAPI_GR_IMG_HANDLE       GraphicImageHandle,
    SAPI_GR_IMG_INFO         *GraphicImageInfo,
    unsigned long             Flags
);
```

### Parameters

#### Handle

[in] Handle of the SAPI session created using [SAPIHandleAcquire](#).

#### GraphicImageHandle

[in] A graphical image handle of the graphical signature image that needs to be updated. This handle was created using one of the following functions: [SAPIGraphicSigImageEnumCont](#), [SAPIGraphicSigImageGUISelect](#), [SAPICertificatesEnumCont](#), or [SAPIGraphicSigImageGetDefault](#).

#### GraphicImageInfo

[in] A pointer to a structure that contains the new graphical image information. The mandatory structure members for this function are: *GraphicImage*, *GraphiImageLen*, *szGraphicImageName*, and *StructLen*. All other members are ignored.

#### Flags

[in] Reserved for future use and must be set to zero.

### Return Values

If the function succeeds, the return value is `SAPI_OK`.

Some possible error codes are listed in the following table.

Value	Meaning
<code>SAPI_ERR_SESSION_IS_NULL</code>	SAPI session handle is NULL, was not created using <a href="#">SAPIHandleAcquire</a> , or is corrupt.
<code>SAPI_ERR_NULL_VALUE_IS_NOT_ALLOWED</code>	<i>GraphicImageHandle</i> or <i>GraphicImageInfo</i> is NULL and must not be NULL.
<code>SAPI_ERR_INVALID_HANDLE</code>	The <i>GraphicImageHandle</i> is corrupt or not initialized correctly,

### Remarks

The name of a graphical signature image may not be NULL and cannot be composed of white characters only.

When updating a graphical signature image, verify that the image format is not damaged. Damage to the image format will prevent CoSign Signature Local from retrieving information such as width and height when [SAPIGraphicSigImageInfoGet](#) is subsequently called, and may cause unexpected results.

***See Also***

[SAPIHandleAcquire](#), [SAPIGraphicSigImageGUISelect](#), [SAPIHandleRelease](#),  
[SAPIGraphicSigImageEnumCont](#), [SAPICertificatesEnumCont](#), [SAPIGraphicSigImageGetDefaultEx](#),  
[SAPIGraphicSigImageInfoUpdate](#), [SAPIGraphicSigLogoInfoUpdate](#)

## SAPIGraphicSigInitialsInfoUpdate

The **SAPIGraphicSigInitialsInfoUpdate** function changes the data of an Initials graphical signature image, its name, or both the data and name. This function updates the image associated with the provided graphical image handle.

```
SAPI int SAPIGraphicSigInitialsInfoUpdate (
    SAPI_SES_HANDLE           Handle,
    SAPI_GR_IMG_HANDLE       GraphicImageHandle,
    SAPI_GR_IMG_INFO         *GraphicImageInfo,
    unsigned long             Flags
);
```

### Parameters

#### Handle

[in] Handle of the SAPI session created using [SAPIHandleAcquire](#).

#### GraphicImageHandle

[in] A graphical image handle of the graphical signature image that needs to be updated. This handle was created using one of the following functions: [SAPIGraphicSigImageEnumCont](#), [SAPIGraphicSigImageGUISelect](#), [SAPICertificatesEnumCont](#), or [SAPIGraphicSigImageGetDefault](#).

#### GraphicImageInfo

[in] A pointer to a structure that contains the new graphical image information. The mandatory structure members for this function are: *GraphicImage*, *GraphiImageLen*, *szGraphicImageName*, and *StructLen*. All other members are ignored.

#### Flags

[in] Reserved for future use and must be set to zero.

### Return Values

If the function succeeds, the return value is `SAPI_OK`.

Some possible error codes are listed in the following table.

Value	Meaning
<code>SAPI_ERR_SESSION_IS_NULL</code>	SAPI session handle is NULL, was not created using <a href="#">SAPIHandleAcquire</a> , or is corrupt.
<code>SAPI_ERR_NULL_VALUE_IS_NOT_ALLOWED</code>	<i>GraphicImageHandle</i> or <i>GraphicImageInfo</i> is NULL and must not be NULL.
<code>SAPI_ERR_INVALID_HANDLE</code>	The <i>GraphicImageHandle</i> is corrupt or not initialized correctly,

### Remarks

The name of a graphical signature image may not be NULL and cannot be composed of white characters only.

When updating a graphical signature image, verify that the image format is not damaged. Damage to the image format will prevent CoSign Signature Local from retrieving information such as width and height when [SAPIGraphicSigImageInfoGet](#) is subsequently called, and may cause unexpected results.

***See Also***

[SAPIHandleAcquire](#), [SAPIGraphicSigImageGUISelect](#), [SAPIHandleRelease](#),  
[SAPIGraphicSigImageEnumCont](#), [SAPICertificatesEnumCont](#), [SAPIGraphicSigImageGetDefaultEx](#),  
[SAPIGraphicSigImageInfoUpdate](#), [SAPIGraphicSigLogoInfoUpdate](#)

## CoSign Signature Local Buffer Signing

CoSign Signature Local can be used to:

Sign files of supported types such as Word and PDF.

To sign a file of the supported type, see [CoSign Signature Local Signature Field](#).

Sign byte streams (raw data) such as records, transactions, or even data that is being read from files of unsupported types.

The functions used for signing byte streams all begin with [SAPIBufferSign](#).

Sign a given hash value. This is intended for cases where the application calculates the hash value of a given data and requests to sign the given hash.

Sign a single string of data (short record, transaction data, etc.). To do so, use [SAPIBufferSign](#).

Starting from CoSign version 5.4, it is possible to generate a PKCS#1 signature in addition to generating a PKCS#7 signature.

If the data is long or fragmented, use the continuous signing functions ([SAPIBufferSignInit](#), [SAPIBufferSignCont](#), and [SAPIBufferSignEnd](#)), as follows: call [SAPIBufferSignInit](#) once to initialize the operation, then call [SAPIBufferSignCont](#) as many times as needed, pointing each time to one portion of the data to be signed. Finally, call the [SAPIBufferSignEnd](#) function to end a signing buffer operation and return the signature data as a PKCS#7 blob or a PKCS#1 signature. Store the signature data in such a way that it can be linked to the original data for verification purposes at a later date.

**Note:** If the signature data is dynamically allocated, call [SAPISigningContextPKCS7BlobLenGet](#) after calling [SAPIBufferSignInit](#), to retrieve the length to be allocated.

## SAPIBufferSignEx

The **SAPIBufferSignEx** function signs data and retrieves the signature data. Use the **SAPIBufferSignEx** function to sign data that can be passed in one call, such as transaction data. If the data is too long for one buffer, or it is fragmented and you wish to sign only certain parts of it, use the trio of functions [SAPIBufferSignInit](#), [SAPIBufferSignCont](#), and [SAPIBufferSignEnd](#).

Use this function also when the *prompt for sign* feature is enabled.

The **SAPIBufferSignEx** function is called once and signs only the data being passed in this call.

```
SAPI int SAPIBufferSignEx (
    SAPI_SES_HANDLE          Handle,
    unsigned char            *Buffer,
    unsigned long            BufferLen,
    unsigned char            *SignedData,
    unsigned long            *SignedDataLen,
    unsigned long            Flags,
    unsigned char            *Credential,
    unsigned long            CredentialLen
);
```

### Parameters

#### Handle

[in] Handle of the SAPI session created by using [SAPIHandleAcquire](#).

#### Buffer

[in] Pointer to the buffer holding the data to be signed.

#### BufferLen

[in] The size in bytes of the data pointed to by *Buffer*.

#### SignedData

[out] Pointer to the buffer holding the signature of the data, represented as a PKCS#7 blob. If the signature data is not required, this parameter can be set to NULL.

If the input *Flags* request the generation of a PKCS#1 signature, *SignedData* will be a PKCS#1 signature.

#### SignedDataLen

[in/out] Pointer to a value specifying the size, in bytes, of the buffer pointed to by the *SignedData* parameter. When the function returns, the value contains the number of bytes actually stored or to be stored in the buffer.

#### Flags

[in] Flag values bit mask. For more information, refer to the following section in Appendix D: [Time-Stamping/OCSP and Miscellaneous Signature-Related Flags](#).

#### Credential

[in] Pointer to a wide char buffer that holds the user credentials required for the signing operation. Contact ARX support for more information. If no special credential is needed, a NULL value can be supplied. However, if the *prompt for sign* feature is enabled, you must supply credentials.

#### CredentialLen

[in] The length in bytes of the content of *Credential*, including NULL separators/terminators.

## Return Values

If the function succeeds, the return value is `SAPI_OK`.

Some possible error codes are listed in the following table.

Value	Meaning
<code>SAPI_ERR_SESSION_IS_NULL</code>	SAPI session handle is NULL, was not created using <a href="#">SAPIHandleAcquire</a> , or is corrupt.
<code>SAPI_ERR_NULL_VALUE_IS_NOT_ALLOWED</code>	<i>Buffer</i> and <i>SignedDataLen</i> are NULL but should not be NULL.
<code>SAPI_ERR_NO_CERT_TO_SELECT_FROM</code>	No certificate was found. This might be due to incorrect credentials, networking problems, or because the preferred certificate could not be found.
<code>SAPI_ERR_TOO_MANY_CERTS_TO_SELECT_FROM</code>	More than one certificate is available, and CoSign Signature Local cannot choose between them. Use either <a href="#">SAPICertificateSetDefault</a> or <a href="#">SAPIConfigurationValueSet</a> to set the certificate with which CoSign Signature Local works.
<code>SAPI_ERR_FAILED_TO_SIGN_BUF</code>	A general error occurred while signing the data. Call <a href="#">SAPIExtendedLastErrorGet</a> for more information about the exact problem.

## Remarks

The certificate that the **SAPIBufferSignEx** function uses for signing is either the one set as the default certificate (for example, by calling the [SAPICertificateSetDefault](#) function), or a certificate that the function automatically chooses. See the [SAPICertificateGetDefault](#) function for more information about the automatic algorithm for choosing a certificate.

To improve performance, if no default certificate is set prior to the signing operation, CoSign Signature Local stores the certificate it used for this operation as the default certificate for the purpose of certifying the next signing operation in the current SAPI session. If this behavior is not required, set the configuration value of [SAPI\\_ENUM\\_CONF\\_ID\\_CERT\\_SET\\_DEFAULT](#) to 0 before the initial call to **SAPIBufferSignEx** in the current session.

In order to generate a PKCS#1 signature, insert the [AR\\_SAPI\\_SIG\\_PKCS1](#) constant into the *Flags* parameter.

In order to specify that the generated signature should be based on a hash mechanism different from SHA-1, insert one of the following flags into the *Flags* parameter: `AR_SAPI_SHA256_FLAG`, `AR_SAPI_SHA384_FLAG`, `AR_SAPI_SHA512_FLAG` or `AR_SAPI_ENFORCE_SHA1_FLAG`. If the given data is a hash value, insert the `AR_SAPI_SIG_HASH_ONLY` constant into the *Flags* parameter.

For signing long data or when the data is fragmented, call the trio of functions [SAPIBufferSignInit](#), [SAPIBufferSignCont](#), and [SAPIBufferSignEndEx](#).

## See Also

[SAPIHandleAcquire](#), [SAPICertificateSetDefault](#), [SAPICertificateGetDefault](#), [SAPIBufferSignInit](#), [SAPIBufferSignCont](#), [SAPIBufferSignEnd](#), [SAPIBufferSign](#), [SAPIConfigurationValueSet](#)

## SAPIBufferSign

The **SAPIBufferSign** function is the older version of the [SAPIBufferSignEx](#) function.

```
SAPI int SAPIBufferSign (  
    SAPI_SES_HANDLE          Handle,  
    unsigned char            *Buffer,  
    unsigned long            BufferLen,  
    unsigned char            *SignedData,  
    unsigned long            *SignedDataLen,  
    unsigned long            Flags  
);
```

## SAPIBufferSignInit

The trio of functions **SAPIBufferSignInit**, [SAPIBufferSignCont](#), and [SAPIBufferSignEnd](#) is used for signing data that is too long for one buffer, or is fragmented and you wish to sign only certain parts of it. The functions are used as follows:

Call the **SAPIBufferSignInit** function to initialize the operation.

Call the [SAPIBufferSignCont](#) function as many times as needed, pointing each time to one portion of the data to be signed.

Call the [SAPIBufferSignEnd](#) function to end a signing buffer operation and return the signature data as a PKCS#7 blob.

```
SAPI int SAPIBufferSignInit (
    SAPI_SES_HANDLE          Handle,
    SAPI_CONTEXT             *SignContext,
    unsigned long            Flags
);
```

### *Parameters*

#### *Handle*

[in] Handle of the SAPI session created by using [SAPIHandleAcquire](#).

#### *SignContext*

[out] Pointer to the Sign Buffer operation context. This context contains internal information about the operation. The information in this context is passed in all subsequent calls to [SAPIBufferSignCont](#) and [SAPIBufferSignEnd](#). Changing the value of *SignContext* by the application is not allowed and can lead to unexpected behavior.

*SignContext* must be released by the

SAPIContextRelease function when the Sign Buffer operation is complete.

### Flags

[in] Flag values bit mask. For more information, refer to the following section in Appendix D: [Time-Stamping/OCSP and Miscellaneous Signature-Related Flags](#).

### Return Values

If the function succeeds, the return value is SAPI\_OK.

Some possible error codes are listed in the following table.

Value	Meaning
SAPI_ERR_SESSION_IS_NULL	SAPI session handle is NULL, was not created using <a href="#">SAPIHandleAcquire</a> , or is corrupt.
SAPI_ERR_NULL_VALUE_IS_NOT_ALLOWED	<i>SignContext</i> is NULL and must not be NULL.
SAPI_FAILED_TO_ALLOCATE_MEMORY	CoSign Signature Local failed to allocate the necessary memory for the new operation context. This might be due to a shortage in system resources.
SAPI_ERR_NO_CERT_TO_SELECT_FROM	No certificate was found. This might be due to incorrect credentials, networking problems, or because the preferred certificate could not be found.
SAPI_ERR_TOO_MANY_CERTS_TO_SELECT_FROM	More than one certificate is available, and CoSign Signature Local cannot choose between them. Use either <a href="#">SAPICertificateSetDefault</a> or <a href="#">SAPIConfigurationValueSet</a> to set the certificate with which CoSign Signature Local works.
SAPI_ERR_FAILED_TO_INIT_SIGN_BUF	A general error occurred while initializing the signing data operation. Call <a href="#">SAPIExtendedLastErrorGet</a> for more information about the exact problem.

### Remarks

The certificate that is used by the signing operation is determined by this function. Once this function is called, any change in the default certificate and any additions of available certificates does not affect this signing operation. Nevertheless, if the selected certificate becomes unavailable before calling the [SAPIBufferSignEnd](#) function, an error is returned by the latter.

The certificate that the **SAPIBufferSignInit** function uses for signing is either the one set as the default certificate (for example by calling the [SAPICertificateSetDefault](#) function), or a certificate that the function automatically chooses. See the [SAPICertificateGetDefault](#) function for more information about the automatic algorithm for choosing a certificate.

To improve performance, when no default certificate is set prior to the signing operation, CoSign Signature Local stores the certificate it used for this operation as the default for the purpose of certifying the next signing operation in the current SAPI session. If this behavior is not required, set the configuration value of

[SAPI\\_ENUM\\_CONF\\_ID\\_CERT\\_SET\\_DEFAULT](#) to 0 before the initial call to [SAPISignatureFieldSign](#) in the current session.

***See Also***

[SAPIHandleAcquire](#), [SAPIBufferSignCont](#), [SAPIBufferSignEnd](#), [SAPICertificateSetDefault](#), [SAPICertificateGetDefault](#), [SAPIBufferSign](#),

SAPIContextRelease, [SAPIConfigurationValueSet](#)

## SAPIBufferSignCont

The **SAPIBufferSignCont** function feeds an additional buffer to the sign buffer operation initiated by [SAPIBufferSignInit](#).

```
SAPI int SAPIBufferSignCont (
    SAPI_SES_HANDLE      Handle,
    SAPI_CONTEXT         *SignContext,
    unsigned char         *Buffer,
    unsigned long         BufferLen
);
```

### ***Parameters***

#### *Handle*

[in] Handle of the SAPI session created by using [SAPIHandleAcquire](#).

#### *SignContext*

[in] Pointer to the Sign Buffer operation context created using [SAPIBufferSignInit](#). *SignContext* must be released by the

SAPIContextRelease function after the signature data is retrieved by [SAPIBufferSignEnd](#), or if the operation is stopped for any reason.

**Buffer**

[in] Pointer to the buffer holding one portion of the data to be signed.

**BufferLen**

[in] Number of bytes of data in the buffer.

**Return Values**

If the function succeeds, the return value is SAPI\_OK.

Some possible error codes are listed in the following table.

Value	Meaning
SAPI_ERR_SESSION_IS_NULL	SAPI session handle is NULL, was not created using <a href="#">SAPIHandleAcquire</a> , or is corrupt.
SAPI_ERR_CONTEXT_NOT_INITIALIZED	The sign buffer operation context was not initialized properly or is corrupt.
SAPI_ERR_NULL_VALUE_IS_NOT_ALLOWED	<i>Buffer</i> is NULL and must not be NULL.
SAPI_ERR_FAILED_TO_CONT_SIGN_BUF	A general error occurred during the course of the signing data operation. Call <a href="#">SAPIExtendedLastErrorGet</a> for more information about the exact problem.

**Remarks**

The sizes of the buffers sent to the **SAPIBufferSignCont** function may differ, but the **order** of the data that is sent to the **SAPIBufferSignCont** function is very important and must be kept when calling the verification functions.

The sizes of the buffers sent when calling verification functions do not need to match the sizes of the buffers sent to the **SAPIBufferSignCont** function, but the order of the bytes must be kept, as well as the total length of the data.

**See Also**

[SAPIHandleAcquire](#), [SAPIBufferSignInit](#), [SAPIBufferSignEnd](#),

SAPIContextRelease

## SAPIBufferSignEndEx

The **SAPIBufferSignEndEx** function ends a signing buffer operation and returns the signature data as a PKCS#7 blob.

Use this function also when the *prompt for sign* feature is enabled.

```
SAPI int SAPIBufferSignEndEx (  
    SAPI_SES_HANDLE          Handle,  
    SAPI_CONTEXT             *SignContext,  
    unsigned char             *SignedData,  
    unsigned long             *SignedDataLen,  
    unsigned char             *Credential,  
    unsigned long             CredentialLen  
);
```

### Parameters

#### Handle

[in] Handle of the SAPI session created by using [SAPIHandleAcquire](#).

#### SignContext

[in] Pointer to the Sign Buffer operation context created using [SAPIBufferSignInit](#).  
*SignContext* must be released by the

SAPIContextRelease function after the [SAPIBufferSignEnd](#) returns.

### *SignedData*

[out] Pointer to the buffer holding the signature of the data, represented as a PKCS#7 blob. If the signature data is not required, this parameter can be set to NULL.

### *SignedDataLen*

[in/out] Pointer to a value specifying the size, in bytes, of the buffer pointed to by the *SignedData* parameter. When the function returns, the value contains the number of bytes actually stored in the buffer.

### *Credential*

[in] Pointer to a wide char buffer that holds the user credentials required for the signing operation. Contact ARX support for more information. If no special credential is needed, a NULL value can be supplied. However, if the *prompt for sign* feature is enabled, you must supply credentials.

### *CredentialLen*

[in] The length in bytes of the content of *Credentials* including NULL separators terminators.

### **Return Values**

If the function succeeds, the return value is SAPI\_OK.

Some possible error codes are listed in the following table.

Value	Meaning
SAPI_ERR_SESSION_IS_NULL	SAPI session handle is NULL, was not created using <a href="#">SAPIHandleAcquire</a> , or is corrupt.
SAPI_ERR_CONTEXT_NOT_INITIALIZED	The sign buffer operation context was not initialized properly or is corrupt.
SAPI_ERR_NULL_VALUE_IS_NOT_ALLOWED	<i>SignedDataLen</i> is NULL and must not be NULL.
SAPI_ERR_FAILED_TO_END_SIGN_BUF	A general error occurred while retrieving the signature data. Call <a href="#">SAPIExtendedLastErrorGet</a> for more information about the exact problem.

### **See Also**

[SAPIHandleAcquire](#), [SAPIBufferSignInit](#), [SAPIBufferSignCont](#), [SAPIBufferSignEnd](#), [SAPISigningContextPKCS7BlobLenGet](#)

## SAPIBufferSignEnd

The **SAPIBufferSignEnd** function is the older version of the [SAPIBufferSignEndEx](#) function.

```
SAPI int SAPIBufferSignEnd (  
    SAPI_SES_HANDLE          Handle,  
    SAPI_CONTEXT             *SignContext,  
    unsigned char            *SignedData,  
    unsigned long            *SignedDataLen  
);
```

## SAPISigningContextPKCS7BlobLenGet

The **SAPISigningContextPKCS7BlobLenGet** function calculates the maximum number of bytes needed for the signature data, if the desired signature is based on PKCS#7. The length that is returned is based on the configuration settings (certificate, signing attributes, etc.) that were used for the signing operation initialization and are referred to by *SignContext*. Note that the length returned by this function is always greater than or equal to the actual number of bytes needed.

This function is called by applications that need to know the size to be allocated for the signature data before performing the whole signing operation.

```
SAPI int SAPISigningContextPKCS7BlobLenGet (  
    SAPI_SES_HANDLE          Handle,  
    SAPI_CONTEXT            *SignContext,  
    unsigned long            *SignatureLen  
);
```

### Parameters

#### Handle

[in] Handle of the SAPI session created by using [SAPIHandleAcquire](#).

#### SignContext

[in] Pointer to the Signing operation context created by the [SAPIBufferSignInit](#) function.

#### SignatureLen

[out] Pointer to a value that holds the maximum number of bytes needed for the signature data.

### Return Values

If the function succeeds, the return value is `SAPI_OK`.

Some possible error codes are listed in the following table.

Value	Meaning
<code>SAPI_ERR_SESSION_IS_NULL</code>	SAPI session handle is NULL, was not created using <a href="#">SAPIHandleAcquire</a> , or is corrupt.
<code>SAPI_ERR_NULL_VALUE_IS_NOT_ALLOWED</code>	<i>SignatureLen</i> is NULL and must not be NULL.
<code>SAPI_ERR_CONTEXT_NOT_INITIALIZED</code>	The buffer signing operation context is either not initialized or is corrupt.

### Remarks

Only call this function after calling the [SAPIBufferSignInit](#) function.

### See Also

[SAPIHandleAcquire](#), [SAPIBufferSignInit](#), [SAPIBufferSignEnd](#)

## CoSign Signature Local - Buffer Verify

CoSign Signature Local can be used to:

Verify files of supported types, such as Word and PDF.

To verify a file of the supported type, see [CoSign Signature Local Signature Field](#).

Verify byte streams, such as records, transactions, or even data that is being read from files of unsupported types.

The functions used for verifying byte streams all begin with [SAPIBufferVerifySignature](#).

Verify a signature based on a given hash value. This is intended for cases where the application calculates the hash value of a given data, and requests to verify a given signature that is based on the given hash.

Verify a single stream of data (short record, transaction data, etc.). To do so, use [SAPIBufferVerifySignature](#).

Starting from CoSign version 5.4, it is possible to validate a PKCS#1 signature in addition to validating a PKCS#7 signature.

If the data is long or fragmented, use the continuous verify functions ([SAPIBufferVerifySignatureInit](#), [SAPIBufferVerifySignatureCont](#) and [SAPIBufferVerifySignatureEnd](#)) as follows: call [SAPIBufferVerifySignatureInit](#) once to initialize the operation, then call [SAPIBufferVerifySignatureCont](#) as many times as needed, pointing each time to one portion of the data to be verified. Finally, call [SAPIBufferVerifySignatureEnd](#) to actually perform the signature verification and check the signing certificate status.

**Note:** Signature verification operations do not require access to the appliance. Moreover, client installation for a machine used only for verification does not require all the client components (see the DocuSign Signature Appliance User Manual for the installation requirements for verify-only mode).

## SAPIBufferVerifySignature

The **SAPIBufferVerifySignature** function verifies a signature of a single buffer. The function optionally returns the certificate validity status. This function is used for verifying short data. For verifying long or fragmented data, use the trio of functions [SAPIBufferVerifySignatureInit](#), [SAPIBufferVerifySignatureCont](#), and [SAPIBufferVerifySignatureEnd](#).

```
SAPI int SAPIBufferVerifySignature (
    SAPI_SES_HANDLE          Handle,
    unsigned char            *Buffer,
    unsigned long            BufferLen,
    unsigned char            *SignedData,
    unsigned long            SignedDataLen,
    SAPI_FILETIME           *SignatureTime,
    SAPI_CERT_STATUS_STRUCT *CertificateStatus,
    unsigned long            Flags
);
```

### Parameters

#### *Handle*

[in] Handle of the SAPI session created by using [SAPIHandleAcquire](#).

#### *Buffer*

[in] Pointer to the buffer that holds the data that was signed.

#### *BufferLen*

[in] Size in bytes of the data in *Buffer*.

#### *SignedData*

[in] Pointer to the buffer that holds the signature value.

#### *SignedDataLen*

[in] Size in bytes of the signature value.

#### *SignatureTime*

[in] The time when the signature was created. This parameter is optional; if this parameter is not required, a NULL value should be provided.

#### *CertificateStatus*

[out] The certificate validity status. This parameter must be supplied.

#### *Flags*

[in] Flag values bit mask. For more information, refer to the following section in Appendix D: [Flags when Verifying a Signature](#).

### Return Values

If the function succeeds, the return value is SAPI\_OK.

Some possible error codes are listed in the following table.

Value	Meaning
SAPI_ERR_SESSION_IS_NULL	SAPI session handle is NULL, was not created using <a href="#">SAPIHandleAcquire</a> , or is corrupt.
SAPI_ERR_NULL_VALUE_IS_NOT_ALLOWED	<i>Buffer</i> and <i>SignedData</i> is NULL and must not be NULL.
SAPI_ERR_FAILED_TO_VERIFY_BUF	A general error occurred while verifying a signature. Call <a href="#">SAPIExtendedLastErrorGet</a> for more information about the exact problem.
SAPI_ERR_SIGNATURE_IS_NOT_VALID	The signature is not valid. The data was probably changed after it was signed.

### Remarks

It is recommended to carefully check the certificate status and make sure its value is as expected. For example, whereas in some environments the inability to check a CRL can be a real problem, in other environments it cannot be checked at all.

To control the attributes to be checked when verifying a certificate, set SAPI\_ENUM\_CONF\_ID\_CERT\_CHAIN\_FLAGS (see [SAPI\\_ENUM\\_CONF\\_ID](#)) to a desired value, using the [SAPIConfigurationValueSet](#) function. Contact ARX for information about which value to pass.

In order to validate a PKCS#1 signature, insert the [AR\\_SAPI\\_SIG\\_PKCS1](#) constant into the *Flags* parameter.

In order to specify that the signature should be based on a hash mechanism different from SHA-1, insert one of the following flags into the *Flags* parameter: AR\_SAPI\_SHA256\_FLAG, AR\_SAPI\_SHA384\_FLAG, AR\_SAPI\_SHA512\_FLAG or AR\_SAPI\_ENFORCE\_SHA1\_FLAG.

If the given data is a hash value, insert the AR\_SAPI\_SIG\_HASH\_ONLY constant into the *Flags* parameter.

### See Also

[SAPIHandleAcquire](#), [SAPIBufferSign](#), [SAPIConfigurationValueSet](#), [SAPIBufferVerifySignatureInit](#), [SAPIBufferVerifySignatureCont](#), [SAPIBufferVerifySignatureEnd](#)

## SAPIBufferVerifySignatureInit

The trio of functions [SAPIBufferVerifySignatureInit](#), [SAPIBufferVerifySignatureCont](#), and [SAPIBufferVerifySignatureEnd](#) is used for verifying data that is too long for one buffer, or is fragmented and only certain parts of it were signed. The functions are used as follows:

Call the [SAPIBufferVerifySignatureInit](#) function to initialize the operation.

Call the [SAPIBufferVerifySignatureCont](#) function as many times as needed, pointing each time to one portion of the data to be verified.

Call the [SAPIBufferVerifySignatureEnd](#) function to actually perform the verification.

```
SAPI int SAPIBufferVerifySignatureInit (  
    SAPI_SES_HANDLE          Handle,  
    SAPI_CONTEXT             *VerifyContext,  
    unsigned char             *SignedData,  
    unsigned long             SignedDataLen,  
    unsigned long             Flags  
);
```

### Parameters

#### *Handle*

[in] Handle of the SAPI session created by using [SAPIHandleAcquire](#).

#### *VerifyContext*

[out] Pointer to the Verify Buffer Signature operation context. This context contains internal information about the operation. The information in this context is passed in all subsequent calls to [SAPIBufferVerifySignatureCont](#) and [SAPIBufferVerifySignatureEnd](#). Changing the value of *VerifyContext* by the application is not allowed and can lead to unexpected behavior. *VerifyContext* must be released by the

SAPIContextRelease function when the Verify Buffer operation is finished.

### *SignedData*

[in] Pointer to the buffer that holds the signature value.

### *SignedDataLen*

[in] Size in bytes of the signature value.

### *Flags*

[in] Flag values bit mask. For more information, refer to the following section in Appendix D: [Flags when Verifying a Signature](#).

### **Return Values**

If the function succeeds, the return value is SAPI\_OK.

Some possible error codes are listed in the following table.

Value	Meaning
SAPI_ERR_SESSION_IS_NULL	SAPI session handle is NULL, was not created using <a href="#">SAPIHandleAcquire</a> , or is corrupt.
SAPI_ERR_NULL_VALUE_IS_NOT_ALLOWED	<i>VerifyContext</i> or <i>SignedData</i> is NULL and must not be NULL.
SAPI_FAILED_TO_ALLOCATE_MEMORY	CoSign Signature Local failed to allocate the necessary memory for the new operation context. This might be due to a shortage in system resources.
SAPI_ERR_FAILED_TO_VERIFY_BUF	A general error occurred while initializing the verify signature operation. Call <a href="#">SAPIExtendedLastErrorGet</a> for more information about the exact problem.

### **Remarks**

To control the attributes to be checked when verifying a certificate, set SAPI\_ENUM\_CONF\_ID\_CERT\_CHAIN\_FLAGS (see [SAPI\\_ENUM\\_CONF\\_ID](#)) to a desired value, using the [SAPIConfigurationValueSet](#) function. Contact ARX for information about which value to pass.

### **See Also**

[SAPIHandleAcquire](#), [SAPIBufferSign](#), [SAPIConfigurationValueSet](#), [SAPIBufferVerifySignatureCont](#), [SAPIBufferVerifySignatureEnd](#)

## SAPIBufferVerifySignatureCont

The **SAPIBufferVerifySignatureCont** function feeds an additional buffer to the Verify Buffer operation.

```
SAPI int SAPIBufferVerifySignatureCont (  
    SAPI_SES_HANDLE      Handle,  
    SAPI_CONTEXT        *VerifyContext,  
    unsigned char        *Buffer,  
    unsigned long        BufferLen  
);
```

### **Parameters**

#### *Handle*

[in] Handle of the SAPI session created by using [SAPIHandleAcquire](#).

#### *VerifyContext*

[in] Pointer to the Verify Buffer Signature operation context created using [SAPIBufferVerifySignatureInit](#).  
*VerifyContext* must be released by the

SAPIContextRelease function after the signature data is verified by [SAPIBufferVerifySignatureEnd](#), or if the operation is stopped for any reason.

**Buffer**

[in] Pointer to the buffer holding one portion of the data that was signed.

**BufferLen**

[in] Size in bytes of the data in *Buffer*.

**Return Values**

If the function succeeds, the return value is SAPI\_OK.

Some possible error codes are listed in the following table.

Value	Meaning
SAPI_ERR_SESSION_IS_NULL	SAPI session handle is NULL, was not created using <a href="#">SAPIHandleAcquire</a> , or is corrupt.
SAPI_ERR_NULL_VALUE_IS_NOT_ALLOWED	<i>Buffer</i> or <i>VerifyContext</i> is NULL and must not be NULL.
SAPI_ERR_FAILED_TO_VERIFY_BUF	A general error occurred during the course of the verifying buffer operation. Call <a href="#">SAPIExtendedLastErrorGet</a> for more information about the exact problem.

**Remarks**

The sizes of the buffers sent to the **SAPIBufferVerifySignatureCont** function may differ, but the **order** of the data that is sent to the **SAPIBufferVerifySignatureCont** function is very important and must be the same as when calling the signing functions.

The sizes of the buffers sent when calling signing functions do not need to match the sizes of the buffers sent to the **SAPIBufferVerifySignatureCont** function, but the order of the bytes must be kept, as well as the total length of the data.

**See Also**

[SAPIHandleAcquire](#), [SAPIBufferSign](#), [SAPIBufferSignCont](#), [SAPIBufferVerifySignatureInit](#), [SAPIBufferVerifySignatureEnd](#)

## SAPIBufferVerifySignatureEnd

The **SAPIBufferVerifySignatureEnd** function ends a verify buffer operation and actually performs the signature verification. The function also optionally returns the certificate validity status.

```
SAPI int SAPIBufferVerifySignatureEnd (  
    SAPI_SES_HANDLE          Handle,  
    SAPI_CONTEXT             *VerifyContext,  
    SAPI_FILETIME            *SignatureTime,  
    SAPI_CERT_STATUS_STRUCT  *CertificateStatus,  
);
```

### *Parameters*

#### *Handle*

[in] Handle of the SAPI session created by using [SAPIHandleAcquire](#).

#### *VerifyContext*

[in] Pointer to the Verify Buffer Signature operation context created using [SAPIBufferVerifySignatureInit](#). *VerifyContext* must be released by

SAPIContextRelease after **SAPIBufferVerifySignatureEnd** returns.

#### *SignatureTime*

[out] The time when the signature was created. Not supported in this version and its value is NULL.

#### *CertificateStatus*

[out] The certificate validity status. This parameter must be supplied.

#### **Return Values**

If the function succeeds, the return value is SAPI\_OK.

Some possible error codes are listed in the following table.

Value	Meaning
SAPI_ERR_SESSION_IS_NULL	SAPI session handle is NULL, was not created using <a href="#">SAPIHandleAcquire</a> , or is corrupt.
SAPI_ERR_NULL_VALUE_IS_NOT_ALLOWED	<i>VerifyContext</i> is NULL and must not be NULL.
SAPI_ERR_FAILED_TO_VERIFY_BUF	A general error occurred while verifying a signature. Call <a href="#">SAPIExtendedLastErrorGet</a> for more information about the exact problem.

#### **Remarks**

It is recommended to carefully check the certificate status and make sure its value is as expected. For example, whereas in some environments the inability to check a CRL can be a real problem, in other environments it cannot be checked at all.

#### **See Also**

[SAPIHandleAcquire](#), [SAPIBufferSign](#), [SAPIBufferVerifySignatureInit](#), [SAPIBufferVerifySignatureCont](#)

## CoSign Signature Local - Enrollment

CoSign Signature Local can be used to:

- Generate a new signature key for a user.

- Create a certificate request for the user. The certificate request can be sent by the user to an external CA.

- Import the certificate that was enrolled by an external CA.

There is no specific function that generates a certificate request. This can be achieved by other third party tools in conjunction with SAPI functionality.

The third party tool will use the public key that is an output parameter of the newly generated signature key. The certificate request that is sent to the CA is based on signing the request with the newly generated key based on a PKCS#1 format.

## SAPIKeyPairGenerate

The **SAPIKeyPairGenerate** function generates a signature key. The function returns the ID (called *ContainerName*) of the newly generated key and also the public key of the newly generated key.

```
SAPI int SAPIKeyPairGenerate (
    SAPI_SES_HANDLE      Handle,
    SAPI_LPCWSTR         UserLoginName,
    SAPI_LPCWSTR         DomainName,
    unsigned char        *Password,
    long                 PasswordLen,
    unsigned long        Flags,
    SAPI_LPCWSTR         ContainerName,
    unsigned long        *ContainerNameLen,
    unsigned char        *PublicKey,
    unsigned long        *PublicKeyLen
);
```

### Parameters

#### *Handle*

[in] Handle of the SAPI session created by using [SAPIHandleAcquire](#).

#### *UserLoginName*

[in] The name under which this session is logged in.

#### *DomainName*

[in] In an active directory environment – the name of the domain to which *UserLoginName* belongs.

In a Directory Independent environment or LDAP environment – this parameter is ignored and its value is NULL.

#### *Password*

[in] The password of *UserLoginName*.

Note that the password value is represented as a wide char string.

#### *PasswordLen*

[in] The length in bytes of the password, including the NULL terminator.

#### *Flags*

[in] Flag values bit mask. For more information, refer to the following section in Appendix D: [Flags used by the SAPILogonEx Function](#).

#### *ContainerName*

[out] Pointer to the buffer that will contain the string identifier of the newly generated key.

#### *ContainerNameLen*

[in/out] Pointer to a value that specifies the size, in bytes, of the data type pointed to by the *ContainerName* parameter. When the function returns, this value contains the size of the data copied to *ContainerName*, or, if *ContainerName* is NULL, the size that needs to be allocated for the *ContainerName* parameter.

#### *PublicKey*

[out] Pointer to the buffer that holds the returned public key value.

### *PublicKeyLen*

[in/out] Pointer to a value that specifies the size, in bytes, of the data type pointed to by the *PublicKey* parameter. When the function returns, this value contains the size of the data copied to *PublicKey*, or, if *PublicKey* is NULL, the size that needs to be allocated for the *PublicKey* parameter.

### **Return Values**

If the function succeeds, the return value is `SAPI_OK`.

Some possible error codes are listed in the following table.

Value	Meaning
<code>SAPI_ERR_SESSION_IS_NULL</code>	SAPI session handle is NULL, was not created using <a href="#">SAPIHandleAcquire</a> , or is corrupt.

### **Remarks**

The public exponent value of the newly generated signature key is always 65537 (i.e., 0x010001).

If the *Flags* input parameter has the `SAPI_KEY_PAIR_4096` value, the newly generated key will be a 4096 bit RSA key. Take into account that if DocuSign Signature Appliance is deployed in FIPS mode, it is not possible to generate a 4096 bit RSA key.

### **See Also**

[SAPIHandleAcquire](#), [SAPIBufferSign](#), [SAPIImportCertificate](#)

## SAPIImportCertificate

The **SAPIImportCertificate** function imports to DocuSign Signature Appliance a certificate which matches a signature key that was previously generated. The function accepts a new certificate that matches the signature key.

```
SAPI int SAPIImportCertificate (
    SAPI_SES_HANDLE      Handle,
    SAPI_LPCWSTR         UserLoginName,
    SAPI_LPCWSTR         DomainName,
    unsigned char        *Password,
    long                 PasswordLen,
    unsigned char        *Certificate,
    unsigned long        CertificateLen
);
```

### Parameters

#### Handle

[in] Handle of the SAPI session created by using [SAPIHandleAcquire](#).

#### UserLoginName

[in] The name under which this session is logged in.

#### DomainName

[in] In an active directory environment – the name of the domain to which *UserLoginName* belongs.

In a Directory Independent environment or LDAP environment – this parameter is ignored and its value is NULL.

#### Password

[in] The password of *UserLoginName*.

Note that the password value is represented as a wide char string.

#### PasswordLen

[in] The length in bytes of the password, including the NULL terminator.

#### Certificate

[in] Pointer to the buffer that will contain the certificate.

#### CertificateLen

[in] The size of the certificate.

### Return Values

If the function succeeds, the return value is `SAPI_OK`.

Some possible error codes are listed in the following table.

Value	Meaning
<code>SAPI_ERR_SESSION_IS_NULL</code>	SAPI session handle is NULL, was not created using <a href="#">SAPIHandleAcquire</a> , or is corrupt.

***See Also***

[SAPIHandleAcquire](#), [SAPIBufferSign](#), [SAPIKeyPairGenerate](#)

## CoSign Signature Local - Obsolete Functions

### **SAPIGraphicSigImageGet**

This function is obsolete. See [Graphical Image Objects](#) for information about new graphical image functions.

### **SAPIGraphicSigImageSet**

This function is obsolete. See [Graphical Image Objects](#) for information about new graphical image functions.



# Signature Local User Management API

The following sections describe in detail the SAPICrypt User Management API.

## Using SAPIUM

SAPIUM refer to the CoSign Signature Local functions for managing users and groups. These functions can be called by applications that perform user management operations such as adding and deleting users, updating groups, and synchronizing the DocuSign Signature Appliance user database with the application's user database. For detailed information on CoSign Signature Local Error Codes, refer to Appendix E: [CoSign Signature Local – Signing/Verifying Error Messages](#) in the SAPICrypt Reference Manual. Most of the SAPIUM functions work only in a Directory Independent environment. There are several functions (such as user information retrieval) that work also when DocuSign Signature Appliance is installed in other environments.

The `SAPIUM.dll` is part of the client installation and is located in the `<Program Files>\ARX\ARX Signature API` directory.

The `SAPIUM.lib` file is part of the SK CD and is located under the `SAPI\lib` directory.

## General SAPIUM functions

Use the general SAPIUM functions as follows:

The [SAPIUMInit](#) function must *always* be the first function to be called, and [SAPIUMFinalize](#) must always be called when all work with SAPIUM is completed. After calling [SAPIUMFinalize](#), the only SAPIUM function that can be called is [SAPIUMInit](#).

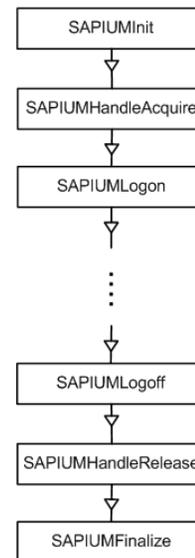
[SAPIUMHandleAcquire](#) should be called before any other function that uses the SAPIUM session handle. [SAPIUMHandleAcquire](#) begins a session that is terminated only by calling [SAPIUMHandleRelease](#) with the session handle as a parameter. An application can have more than one SAPIUM session handle opened with SAPIUM, but since the SAPIUM API is used by administrative and usually interactive applications, the common application is single-threaded.

The [SAPIUMLogonEx](#) function logs into DocuSign Signature Appliance under the provided credentials. This function should be called right after calling [SAPIUMHandleAcquire](#). Since all the SAPIUM functions except [SAPIUMSCPSset](#) must be called by a privileged user, you must call [SAPIUMLogonEx](#) prior to calling these functions.

The [SAPIUMLogoff](#) function ends the session with DocuSign Signature Appliance under a specific user's credentials. The [SAPIUMLogoff](#) function should be called only if the [SAPIUMLogonEx](#) function was previously called, and after ending all work with DocuSign Signature Appliance under the specific user credentials.

A typical single-threaded SAPIUM application will begin by calling [SAPIUMInit](#), followed by a call to [SAPIUMHandleAcquire](#) and then to [SAPIUMLogonEx](#). It will end with calling [SAPIUMLogoff](#), followed by calls to [SAPIUMHandleRelease](#) and [SAPIUMFinalize](#).

An application can call [SAPIUMLibInfoGet](#) to find out the SAPIUM version, and thus make use of specific functionality that is implemented only in new versions.



All the non-void SAPIUM functions return `SAPI_OK` if they succeed and a SAPIUM error code in case of failure. Although the error code usually provides information about which operation failed and why, it is possible and recommended to call [SAPIUMExtendedLastErrorGet](#) for additional information about the reason for failure.

In a high availability environment, client applications can use the [SAPIUMGetTokenID](#) and [SAPIUMSetTokenID](#) to get information about the currently used Appliance or set to which appliance to connect.

The TokenIDs in these functions are compatible with the TokenIDs in the [SAPIGetTokenID](#) and [SAPISetTokenID](#) functions in SAPICrypt.

## Functions for Managing a Single User

The [SAPIUMUserAddEx1](#) function adds a user to DocuSign Signature Appliance. Adding a user automatically triggers the generation of the user's key and certificate.

The [SAPIUMUserUpdate](#) function updates user information in DocuSign Signature Appliance. The changes may trigger the generation of a new user key and certificate for the user.

The [SAPIUMCredentialSet](#) function sets the credentials of a specific user.

The [SAPIUMUserSetLogonState](#) function sets the login status (enabled or disabled) of the given DocuSign Signature Appliance user.

The [SAPIUMCounterReset](#) function resets a signature counter of the user.

The [SAPIUMUserAssignGroup](#) function sets the group of a given user.

The [SAPIUMUserDelete](#) function deletes a user as well as all of the user's keys, certificates, and data objects (e.g., graphical image) from DocuSign Signature Appliance.

The [SAPIUMUserInfoGetEx](#) function retrieves all the information stored in DocuSign Signature Appliance for a specific user.

The [SAPIUMUserGetByLoginName](#) function retrieves the user's handle based on a given user login name.

The [SAPIUMUserTechIDGet](#) function retrieves the internal technical identity of a given user.

The [SAPIUMGroupGetByUserGUID](#) function retrieves the group's record of the group to which a user with a given GUID identity belongs.

## Functions for Managing a Single Group

The [SAPIUMGroupAdd](#) function adds a group to DocuSign Signature Appliance.

The [SAPIUMGroupSetStatusByTechID](#) function sets the state of the group to either Enabled or Disabled. If the group is disabled, all users that are members of the group cannot sign. However, if a user is disabled, then even if his/her group is enabled, the user will nevertheless not be able to login.

The [SAPIUMGroupUpdate](#) function updates the group record inside DocuSign Signature Appliance.

The [SAPIUMGroupDelete](#) function deletes a group from DocuSign Signature Appliance.

The [SAPIUMGroupGetByTechID](#) function retrieves all the group information stored in DocuSign Signature Appliance for a specific group, specified by its TechID.

The [SAPIUMGroupGetByName](#) function retrieves all the group information stored in DocuSign Signature Appliance for a specific group, specified by its group name.

The [SAPIUMGroupExtDataUpdate](#) function updates the extended information of a group. The [SAPIUMGroupExtDataUpdate](#) retrieve an extended information of a group.

## Functions for Synchronizing the Users Database with the Application's Users Database

Applications working with their own users database that wish to synchronize their database with DocuSign Signature Appliance, generally call the SAPIUM add-user and delete-user functions, as needed. However, there are cases in which these actions are not enough to ensure synchronization. For example, in the case of networking problems, or if the application's users database is managed by various applications, some of which do not work with DocuSign Signature Appliance.

For all cases in which user information in DocuSign Signature Appliance differs from user information in the application, SAPIUM offers the ability to enforce synchronization between the two databases using a simple synchronization mechanism. The mechanism works as follows:

1. The application calls [SAPIUMUsersSyncBegin](#) and retrieves a Synchronization Context. The [SAPIUMUsersSyncBegin](#) function indicates to DocuSign Signature Appliance that a synchronization process has begun, and all subsequent calls to the synchronization functions must use the Synchronization Context returned.
2. For **every user** in the application's database, the application calls [SAPIUMUserSync](#) with all the user's information. If the user does not exist in DocuSign Signature Appliance, the user, as well as associated keys and certificate are created. If the user exists, DocuSign Signature Appliance compares the new information with its own, and updates its database and regenerates a new certificate, if necessary.
3. The application calls [SAPIUMUsersSyncEnd](#) to indicate that all users' information has been sent to DocuSign Signature Appliance. When this function is called, DocuSign Signature Appliance scans its entire users database, and deletes all users not referred to by [SAPIUMUserSync](#), as well as their keys and certificates. It is imperative therefore to call [SAPIUMUserSync](#) for **every** user you wish to leave in DocuSign Signature Appliance, even if the user's information in DocuSign Signature Appliance is identical to the information in the application's database.

The application can call the [SAPIUMUsersSyncStop](#) function at any stage to stop the synchronization process. Any subsequent calls to [SAPIUMUserSync](#) and [SAPIUMUsersSyncEnd](#) return an error, and do not affect the DocuSign Signature Appliance user database.

Note that calling [SAPIUMUsersSyncStop](#) is equivalent to stopping the synchronization operation by discontinuing calls to [SAPIUMUserSync](#) for more users and refraining from calling [SAPIUMUsersSyncEnd](#), which indicates the end of the synchronization process.

## Functions for Enumerating All the Users Defined in DocuSign Signature Appliance

To retrieve all users defined in DocuSign Signature Appliance, use [SAPIUMUsersEnumInitEx](#) followed by repetitive calls to [SAPIUMUsersEnumCont](#), to return all the user handles one by one. To retrieve detailed information about a specific user, use [SAPIUMUserInfoGetEx](#).

## Functions for Enumerating All the Groups Defined in DocuSign Signature Appliance

To retrieve all users defined in DocuSign Signature Appliance, use [SAPIUMGroupsEnumInit](#) followed by repetitive calls to [SAPIUMGroupsEnumCont](#), to return all the groups records.

## Functions for Setting Configuration

The [SAPIUMSCPSet](#) function sets the DocuSign Signature Appliance address in the current machine settings. The client on this machine will use this address whenever it needs to connect to DocuSign Signature Appliance.

## SAPIUMInit

The **SAPIUMInit** function initializes the SAPIUM library and must be called before any other function in the library.

```
SAPIUM int SAPIUMInit ();
```

### Parameters

This function has no parameters.

### Return Values

If the function succeeds, the return value is `SAPI_OK`.

Some possible error codes are listed in the following table.

Value	Meaning
<code>SAPI_ERR_FAILED_TO_INIT_REG_KEY</code>	Could not find the install path of the SAPIUM DLL in the registry. A possible reason is CoSign Signature Local is not installed.
<code>SAPI_ERR_FAILED_TO_INIT_LOAD_MODULE</code>	Failed to load the SAPIUM DLL. A possible reason is that the SAPIUM DLL was moved from the location specified during CoSign Signature Local installation.

### Remarks

When no additional SAPIUM function needs to be called, you should call [SAPIUMFinalize](#) in order to close the library and free the library resources.

### See Also

[SAPIUMFinalize](#)

## SAPIUMHandleAcquire

The **SAPIUMHandleAcquire** function acquires a SAPIUM session handle. This returned handle should be used to make calls to all SAPIUM functions that require a session handle.

```
SAPIUM int SAPIUMHandleAcquire (  
    SAPI_UM_SES_HANDLE      *Handle  
);
```

### Parameters

*Handle*

[out] Pointer to a handle of a SAPIUM session.

### Return Values

If the function succeeds, the return value is SAPI\_OK.

Some possible error codes are listed in the following table.

Value	Meaning
SAPI_ERR_NULL_VALUE_IS_NOT_ALLOWED	NULL value for <i>Handle</i> is not allowed.
SAPI_ERR_FAILED_TO_FIND_SERVER	A general error occurred while trying to create a new session handle. Call <a href="#">SAPIUMExtendedLastErrorGet</a> for more information about the exact problem.

### Remarks

When the handle is not required anymore, it should be released by calling the [SAPIUMHandleRelease](#) function. Releasing the session handle frees all the resources related to the handle, and the handle becomes invalid.

### See Also

[SAPIUMHandleRelease](#)

## SAPIUMLogonEx

The **SAPIUMLogonEx** function logs into DocuSign Signature Appliance under the provided credentials. This function should be called right after calling [SAPIUMHandleAcquire](#).

```
SAPIUM int SAPIUMLogonEx (
    SAPI_UM_SES_HANDLE      Handle,
    SAPI_LPCWSTR            UserLoginName,
    SAPI_LPCWSTR            DomainName,
    unsigned char           *Password,
    long                    PasswordLen,
    unsigned long           Flags
);
```

### Parameters

#### Handle

[in] Handle of the SAPIUM session. This handle must have been created using [SAPIUMHandleAcquire](#).

#### UserLoginName

[in] The name under which this session is logged in.

#### DomainName

[in] This parameter is ignored and should be NULL.

#### Password

[in] The password of UserLoginName.

Note that the password value is represented as a wide char string.

#### PasswordLen

[in] The length in bytes of the password, including the NULL terminator.

#### Flags

[in] A bit mask value of the user's rights. See [SAPI\\_UM\\_ENUM\\_LOGON\\_FLAG\\_TYPE](#) for possible values.

### Return Values

If the function succeeds, the return value is `SAPI_OK`.

Some possible error codes are listed in the following table.

Value	Meaning
<code>SAPI_ERR_SESSION_IS_NULL</code>	SAPIUM session handle is NULL, was not created using <a href="#">SAPIUMHandleAcquire</a> , or is corrupt.
<code>SAPI_ERR_ILLEGAL_DIRECTORY_KIND</code>	DocuSign Signature Appliance is not installed in Directory Independent mode.
<code>SAPI_ERR_HANDLE_ALREADY_LOGGED_ON</code>	The session is already logged on. The reason might be that the SAPIUMLogon function was previously called, and no call to <a href="#">SAPIUMLogoff</a> was made.

Value	Meaning
SAPI_ERR_FAILED_TO_LOGON	An error occurred while trying to logon to DocuSign Signature Appliance. Call <a href="#">SAPIUMExtendedLastErrorGet</a> for more information about the exact problem.

**Remarks**

Call [SAPIUMLogoff](#) when there are no other operations that need to be done under the provided user's credentials.

Call **SAPIUMLogonEx** before calling any function that needs to be run under special user privileges.

**See Also**

[SAPIUMHandleAcquire](#), [SAPIUMLogoff](#)

## SAPIUMLogon

The **SAPIUMLogon** function is the older version of the [SAPIUMLogonEx](#) function.

```
SAPI int SAPILogon (  
    SAPI_SES_HANDLE          Handle,  
    SAPI_LPCWSTR             UserLoginName,  
    SAPI_LPCWSTR             DomainName,  
    unsigned char            *Password,  
    long                     PasswordLen,  
);
```

## SAPIUMGetTokenID

The **SAPIUMGetTokenID** function retrieves a Token ID that identifies the Appliance currently used by the SAPIUM Session. This enables more control over the flow of operations. The function is relevant for cases where several Primary appliances are used and the application would like to control and specify which appliance to use for the digital signature operations.

```
SAPI int SAPIGetTokenID (
    SAPI_UM_SES_HANDLE      Handle,
    char                    *TokenID);
```

### Parameters

#### Handle

[in] Handle of the SAPIUM session created by using [SAPIHandleAcquire](#).

#### TokenID

[out] The identification of the appliance used by this SAPIUM Session.

### Return Values

If the function succeeds, the return value is SAPI\_OK.

Some possible error codes are listed in the following table.

Value	Meaning
SAPI_ERR_SESSION_IS_NULL	SAPIUM session handle is NULL, was not created using <a href="#">SAPIUMHandleAcquire</a> , or is corrupt.
SAPI_ERR_ILLEGAL_DIRECTORY_KIND	DocuSign Signature Appliance is not installed in Directory Independent mode.

### See Also

[SAPIUMHandleAcquire](#), [SAPIUMSetTokenID](#)

## SAPIUMSetTokenID

The **SAPIUMSetTokenID** function sets the Token ID that defines which Appliance should be used by the SAPIUM Session. This enables more control over the flow of operations. This function is relevant for cases where several Primary appliances are used and the application would like to control and specify which appliance to use for the digital signature operations.

The command should be called before the SAPIUMLogon command.

```
SAPI int SAPIUMSetTokenID (
    SAPI_UM_SES_HANDLE      Handle,
    char                    *TokenID);
```

### Parameters

#### Handle

[in] Handle of the SAPIUM session created by using [SAPIHandleAcquire](#).

#### TokenID

[in] The identification of the appliance that should be used by this SAPIUM Session.

### Return Values

If the function succeeds, the return value is SAPI\_OK.

Some possible error codes are listed in the following table.

Value	Meaning
SAPI_ERR_SESSION_IS_NULL	SAPIUM session handle is NULL, was not created using <a href="#">SAPIUMHandleAcquire</a> , or is corrupt.
SAPI_ERR_ILLEGAL_DIRECTORY_KIND	DocuSign Signature Appliance is not installed in Directory Independent mode.

### See Also

[SAPIUMHandleAcquire](#),

SAPIUMGetTokenID

## **SAPIUMLogonBuiltIn**

This function is obsolete.

## SAPIUMLogoff

The **SAPIUMLogoff** function ends the session with DocuSign Signature Appliance under a specific user's credentials. The **SAPIUMLogoff** function should be called only if the [SAPIUMLogonEx](#) function was previously called, and after ending all work with DocuSign Signature Appliance under the specific user credentials.

```
SAPIUM int SAPIUMLogoff (
    SAPI_UM_SES_HANDLE          Handle
);
```

### *Parameters*

*Handle*

[in] Handle of the SAPIUM session. This handle must have been created using [SAPIUMHandleAcquire](#) and is the same handle used for the [SAPIUMLogonEx](#) function call.

### *Return Values*

If the function succeeds, the return value is SAPI\_OK.

### *Remarks*

It is recommended to call the [SAPIUMHandleRelease](#) function right after calling the **SAPIUMLogoff** function, although it is not mandatory.

Calling the **SAPIUMLogoff** function does not automatically release the handles that were allocated by SAPIUM functions. These resources must be released by [SAPIUMHandleRelease](#).

### *See Also*

[SAPIUMHandleAcquire](#), [SAPIUMHandleRelease](#), [SAPIUMLogonEx](#)

## SAPIUMHandleRelease

The **SAPIUMHandleRelease** function releases all the resources related to any kind of SAPIUM handles. This function should be called whenever the application finishes using a handle, to avoid unnecessary allocation of resources.

```
SAPIUM void SAPIHandleRelease (  
    Void          *Handle  
);
```

### *Parameters*

*Handle*

[in] Any handle that was created by any of the SAPIUM handle creation functions, such as [SAPIUMHandleAcquire](#), [SAPIUMUsersEnumCont](#).

### *Return Values*

This function has no return values.

### *Remarks*

Using many handles without calling the **SAPIUMHandleRelease** function can lead to a shortage of system resources, and in some cases prevent SAPIUM from allocating new resources.

After calling the **SAPIUMHandleRelease** function, you may not use the handle anymore, and it is recommended to set its value to NULL.

### *See Also*

[SAPIUMHandleAcquire](#), [SAPIUMUsersEnumCont](#)

## **SAPIUMFinalize**

The **SAPIUMFinalize** function ends the work with the SAPIUM library, and releases all the SAPIUM global resources. The **SAPIUMFinalize** function should be called when there are no more calls to SAPIUM functions. Once the **SAPIUMFinalize** function is called, a call to any of the SAPIUM functions can lead to unexpected results.

```
SAPIUM int SAPIUMFinalize ();
```

### ***Parameters***

This function has no parameters.

### ***Return Values***

If the function succeeds, the return value is SAPI\_OK.

### ***See Also***

[SAPIUMInit](#)

## **SAPIUMExtendedLastErrorGet**

The **SAPIUMExtendedLastErrorGet** function retrieves the calling application's extended last-error code value. The Extended last-error is often, although not always, different from the error code that was returned in the last function call, and is useful for getting more information about what caused the last function to fail.

```
SAPIUM int SAPIUMExtendedLastErrorGet();
```

### ***Parameters***

This function has no parameters.

### ***Return Values***

The return value is the calling application's last-error code value. Internal SAPIUM functions set this value whenever an error occurs.

### ***Remarks***

For a complete list of error codes, see the `SAPIUMErrors.h` file in the `SAPI\include` folder in the installation CD.

Call the **SAPIUMExtendedLastErrorGet** function immediately when a function's return value indicates that such a call will return useful data. Immediacy is important because subsequent function calls can reset this value, wiping out the error code set by the most recently failed function.

## SAPIUMLibInfoGet

The **SAPIUMLibInfoGet** function returns some general information about the library.

```
SAPIUM int SAPILibGetInfo(  
    PSAPI_UM_INFO_STRUCT          SAPIUMInfoStruct  
);
```

### **Parameters**

*SAPIUMInfoStruct*

[out] A pointer to the [SAPI\\_UM\\_INFO\\_STRUCT](#) structure that holds the library version information.

### **Return Values**

If the function succeeds, the return value is SAPI\_OK.

## SAPIUMUserAddEx1

The **SAPIUMUserAddEx1** function adds a user to DocuSign Signature Appliance. Adding a user automatically triggers the generation of the user's keys and certificate. This function is relevant only to a Directory Independent environment.

```
SAPIUM int SAPIUMUserAddEx1 (  
    SAPI_UM_SES_HANDLE          Handle,  
    SAPI_LPCWSTR                UserLoginName,  
    SAPI_LPCWSTR                UserCN,  
    SAPI_LPCWSTR                emailAddress,  
    unsigned char               *Password,  
    unsigned long               PasswordLen,  
    SAPI_CALC_CREDENTIALS_CALLBACK CalcCredentialFunc,  
    SAPI_TECH_ID                GroupTechID,  
    unsigned long               Flags  
);
```

### **Parameters**

#### *Handle*

[in] Handle of the SAPIUM session. This handle must have been created using [SAPIUMHandleAcquire](#).

#### *UserLoginName*

[in] The name that will be used for logging into DocuSign Signature Appliance. This field is mandatory. The maximum length of the parameter is 64 characters.

#### *UserCN*

[in] The user's common name. This name will be the subject (issued to) of the certificate. This field is mandatory.

The maximum length of the parameter is 256 characters.

#### *emailAddress*

[in] The user's email address. This value will be part of the subject of the certificate. The maximum length of the parameter is 128 characters.

#### *Password*

[in] The user's password. This field is mandatory.

Note that the password value is represented as a wide char string.

The maximum length of the parameter is 64 characters.

#### *PasswordLen*

[in] The length in bytes of the password, including the NULL terminator.

#### *CalcCredentialFunc*

[in] A pointer to a user-defined callback function that gets the user password and returns the user credentials. Currently this parameter is not supported, and its value should be NULL.

#### *GroupTechID*

[in] The technical identity of a group to which the user belongs. If the user does not belong to any group, the given GroupTechID should be 0.

### Flags

[in] A bit mask value of the user's rights. See [Appendix I: CoSign Signature Local – User Management Constants](#) for possible values.

### Return Values

If the function succeeds, the return value is `SAPI_OK`.

Some possible error codes are listed in the following table.

Value	Meaning
<code>SAPI_ERR_SESSION_IS_NULL</code>	SAPIUM session handle is NULL, was not created using <a href="#">SAPIUMHandleAcquire</a> , or is corrupt.
<code>SAPI_ERR_NULL_VALUE_IS_NOT_ALLOWED</code>	<i>UserLoginName</i> , <i>UserCN</i> , and <i>Password</i> should not be NULL.
<code>SAPI_ERR_FAILED_TO_WRITE_USER</code>	An error occurred while trying to add a new user. Call <a href="#">SAPIUMExtendedLastErrorGet</a> for more information about the exact problem.

### Remarks

The user under whose credentials this operation is performed must have user admin privileges.

If a user with the same user login name already exists in DocuSign Signature Appliance, an error code is returned.

DocuSign Signature Appliance does not store the clear user password but rather a hash value of the password and a unique salt generated by the function, to prevent similar passwords from having similar credential hash values.

### See Also

[SAPIUMHandleAcquire](#), [SAPIUMUserDelete](#), [SAPIUMUserSync](#), [SAPIUMCredentialSet](#)

## SAPIUMUserAdd

The **SAPIUMUserAdd** is the older version of the [SAPIUMUserAddEx1](#) function, dating to before groups were introduced.

```
SAPIUM int SAPIUMUserAdd (
    SAPI_UM_SES_HANDLE          Handle,
    SAPI_LPCWSTR                UserLoginName,
    SAPI_LPCWSTR                UserCN,
    SAPI_LPCWSTR                emailAddress,
    unsigned char               *Password,
    unsigned long               PasswordLen,
    SAPI_CALC_CREDENTIALS_CALLBACK CalcCredentialFunc
    unsigned long               Flags
);
```

## SAPIUMUserUpdate

The **SAPIUMUserUpdate** function updates the user common name, email address and rights. This function is relevant only to a Directory Independent environment.

```
SAPIUM int SAPIUMUserUpdate (
    SAPI_UM_SES_HANDLE      Handle,
    SAPI_LPCWSTR            UserLoginName,
    SAPI_LPCWSTR            UserCN,
    SAPI_LPCWSTR            emailAddress,
    unsigned long           Flags
);
```

### **Parameters**

#### *Handle*

[in] Handle of the SAPIUM session. This handle must have been created using [SAPIUMHandleAcquire](#).

#### *UserLoginName*

[in] The name that will be used for logging into DocuSign Signature Appliance.

#### *UserCN*

[in] The user's common name. This name will be the subject (issued to) of the certificate.

#### *emailAddress*

[in] The user's email address. This value will be part of the subject of the certificate.

#### *Flags*

[in] A bit mask value of the user's rights. See [Appendix I: CoSign Signature Local – User Management Constants](#) for possible values.

### **Return Values**

If the function succeeds, the return value is SAPI\_OK.

### **Remarks**

If the provided UserLoginName does not exist, then a new user will be created with the provided input parameters.

### **See Also**

[SAPIUMHandleAcquire](#), [SAPIUMUserAdd](#), [SAPIUMUserSync](#), [SAPIUMCredentialSet](#)

## SAPIUMCredentialSet

The **SAPIUMCredentialSet** function sets the credentials of a specific user. DocuSign Signature Appliance does not store the clear user password but rather a hash value of the password and a unique salt generated by the function, to prevent similar passwords from having similar credential hash values.

This function is relevant only to a Directory Independent environment.

```
SAPIUM int SAPIUMCredentialSet (
    SAPI_UM_SES_HANDLE          Handle,
    SAPI_LPCWSTR                UserLoginName,
    unsigned char               *Password,
    unsigned long                PasswordLen,
    SAPI_CALC_CREDENTIALS_CALLBACK CalcCredentialFunc
);
```

### Parameters

#### Handle

[in] Handle of the SAPIUM session. This handle must have been created using [SAPIUMHandleAcquire](#).

#### UserLoginName

[in] The login name of the user whose credentials are being set.

#### Password

[in] The user's new password.

Note that the password value is represented as a wide char string.

#### PasswordLen

[in] The length in bytes of the new password, including the NULL terminator.

#### CalcCredentialFunc

[in] A pointer to a user-defined callback function that gets the new password and returns the user credentials. Currently this parameter is not supported, and its value should be NULL.

### Return Values

If the function succeeds, the return value is SAPI\_OK.

Some possible error codes are listed in the following table.

Value	Meaning
SAPI_ERR_SESSION_IS_NULL	SAPIUM session handle is NULL, was not created using <a href="#">SAPIUMHandleAcquire</a> , or is corrupt.
SAPI_ERR_FAILED_TO_UPDATE_CREDENTIAL	An error occurred while trying to set the user password. Call <a href="#">SAPIUMExtendedLastErrorGet</a> for more information about the exact problem.

### Remarks

The user under whose credentials this operation is performed must have user admin privileges.

***See Also***

[SAPIUMHandleAcquire](#), [SAPIUMUserAdd](#)

## SAPIUMUserSetLogonState

The **SAPIUMUserSetLogonState** function sets the status of the user to either Disabled or Enabled. If a user is disabled, the user will not be able to sign even if the user belongs to an enabled group. This function is relevant only to a Directory Independent environment.

**Note:** You can retrieve a user's logon state by using the [SAPIUMUserInfoGetEx](#) function.

```
SAPIUM int SAPIUMUserSetLogonState (
    SAPI_UM_SES_HANDLE          Handle,
    SAPI_LPCWSTR                UserLoginName,
    SAPI_UM_ENUM_USER_LOGIN_STATUS UserLogonStatus
);
```

### Parameters

#### *Handle*

[in] Handle of the SAPIUM session. This handle must have been created using [SAPIUMHandleAcquire](#).

#### *UserLoginName*

[in] The login name of the user.

#### *UserLogonStatus*

[in] The new login status of the user. For information, refer to [SAPI\\_UM\\_ENUM\\_USER\\_LOGIN\\_STATUS](#).

### Return Values

If the function succeeds, the return value is SAPI\_OK.

### See Also

[SAPIUMHandleAcquire](#), [SAPIUMUserAdd](#)

## SAPIUMUserDelete

The **SAPIUMUserDelete** deletes a user as well as his keys, certificates and data objects (e.g., graphical image) from DocuSign Signature Appliance.

This function is relevant only to a Directory Independent environment.

```
SAPIUM int SAPIUMUserDelete (  
    SAPI_UM_SES_HANDLE      Handle,  
    SAPI_LPCWSTR            UserLoginName  
);
```

### Parameters

#### Handle

[in] Handle of the SAPIUM session. This handle must have been created using [SAPIUMHandleAcquire](#).

#### UserLoginName

[in] The login name of the deleted user.

### Return Values

If the function succeeds, the return value is SAPI\_OK.

Some possible error codes are listed in the following table.

Value	Meaning
SAPI_ERR_SESSION_IS_NULL	SAPIUM session handle is NULL, was not created using <a href="#">SAPIUMHandleAcquire</a> , or is corrupt.
SAPI_ERR_NULL_VALUE_IS_NOT_ALLOWED	<i>UserLoginName</i> should not be NULL.
SAPI_ERR_FAILED_TO_DELETE_USER	An error occurred while trying to delete the user. Call <a href="#">SAPIUMExtendedLastErrorGet</a> for more information about the exact problem.

### Remarks

The user under whose credentials this operation is performed must have user admin privileges.

### See Also

[SAPIUMHandleAcquire](#), [SAPIUMUserAdd](#)

## SAPIUMUsersSyncBegin

The **SAPIUMUsersSyncBegin** indicates to DocuSign Signature Appliance that the synchronization between DocuSign Signature Appliance users database and the application users database has begun. The function resets all the users' update time, so that when the [SAPIUMUsersSyncEnd](#) function is called, all users whose update time is still 0 will be removed.

This function is relevant only to a Directory Independent environment.

```
SAPIUM int SAPIUMUsersSyncBegin (
    SAPI_UM_SES_HANDLE      Handle,
    SAPI_UM_CONTEXT         SyncContext
);
```

### Parameters

#### Handle

[in] Handle of the SAPIUM session. This handle must have been created using [SAPIUMHandleAcquire](#).

#### SyncContext

[in] Pointer to the Users Sync operation context (refer to [SAPI\\_UM\\_CONTEXT](#)). This context contains internal information about the operation and should be passed to [SAPIUMUsersSyncEnd](#) and [SAPIUMUsersSyncStop](#). Changing the value of *SyncContext* by the application is not allowed and can lead to unexpected behavior.

### Return Values

If the function succeeds, the return value is SAPI\_OK.

Some possible error codes are listed in the following table.

Value	Meaning
SAPI_ERR_SESSION_IS_NULL	SAPIUM session handle is NULL, was not created using <a href="#">SAPIUMHandleAcquire</a> , or is corrupt.
SAPI_ERR_NULL_VALUE_IS_NOT_ALLOWED	<i>SyncContext</i> should not be NULL.
SAPI_ERR_FAILED_TO_BEGIN_SYNC	An error occurred while trying to begin the users database synchronization process. Call <a href="#">SAPIUMExtendedLastErrorGet</a> for more information about the exact problem.

### Remarks

The user under whose credentials this operation is performed must have user admin privileges.

### See Also

[SAPIUMHandleAcquire](#), [SAPIUMUsersSyncStop](#), [SAPIUMUsersSyncEnd](#), [SAPIUMUserSync](#)

## SAPIUMUserSync

The **SAPIUMUserSync** function synchronizes the DocuSign Signature Appliance users database and the application's users database for single user information. This function should be called for every user in the application database, after calling [SAPIUMUsersSyncBegin](#) and before calling [SAPIUMUsersSyncEnd](#). This function is relevant only to a Directory Independent environment.

The results of calling **SAPIUMUserSync** are as follows:

If the user information is new for DocuSign Signature Appliance, the user and his keys and certificate will be created.

If the user information is different from the information stored in DocuSign Signature Appliance, the information will be updated in DocuSign Signature Appliance, and, if necessary, a new certificate will be created.

If the user information is identical to the information in DocuSign Signature Appliance, no action is taken.

In all cases, the update time of the user record in DocuSign Signature Appliance is changed to the current time.

```
SAPIUM int SAPIUMUserSync (
    SAPI_UM_SES_HANDLE          Handle,
    SAPI_LPCWSTR                UserLoginName,
    SAPI_LPCWSTR                UserCN,
    SAPI_LPCWSTR                emailAddress,
    unsigned char               *Password,
    unsigned long               PasswordLen,
    SAPI_CALC_CREDENTIALS_CALLBACK CalcCredentialFunc
    unsigned long               Flags
);
```

### Parameters

#### *Handle*

[in] Handle of the SAPIUM session. This handle must have been created using [SAPIUMHandleAcquire](#).

#### *UserLoginName*

[in] The name that will be used for logging into DocuSign Signature Appliance.

#### *UserCN*

[in] The user's common name. This name will be the subject (issued to) of the certificate.

#### *emailAddress*

[in] The user's email address. This value will be part of the subject of the certificate.

#### *Password*

[in] The user's password.

Note that the password value is represented as a wide char string.

#### *PasswordLen*

[in] The length in bytes of the password, including the NULL terminator.

### *CalcCredentialFunc*

[in] A pointer to a user defined callback function that gets the new password and returns the user credentials. Currently this parameter is not supported, and its value should be NULL.

### *Flags*

[in] A bit mask value of the user's rights. See [Appendix I: CoSign Signature Local – User Management Constants](#) for possible values.

### **Return Values**

If the function succeeds, the return value is SAPI\_OK.

Some possible error codes are listed in the following table.

Value	Meaning
SAPI_ERR_SESSION_IS_NULL	SAPIUM session handle is NULL, was not created using <a href="#">SAPIUMHandleAcquire</a> , or is corrupt.
SAPI_ERR_NULL_VALUE_IS_NOT_ALLOWED	<i>UserLoginName</i> , <i>UserCN</i> , and <i>Password</i> , should not be NULL.
SAPI_ERR_FAILED_TO_WRITE_USER	An error occurred while trying to add a new user. Call <a href="#">SAPIUMExtendedLastErrorGet</a> for more information about the exact problem.

### **Remarks**

The user under whose credentials this operation is performed must have user admin privileges.

Changes caused by this function to the users database cannot be reversed by calling [SAPIUMUsersSyncStop](#).

### **See Also**

[SAPIUMHandleAcquire](#), [SAPIUMUsersSyncBegin](#), [SAPIUMUsersSyncStop](#), [SAPIUMUsersSyncEnd](#)

## SAPIUMUsersSyncEnd

The **SAPIUMUsersSyncEnd** function indicates to DocuSign Signature Appliance that the synchronization between users database and the application users database has ended. The function scans the entire users database and finds users whose update time is 0 (the value given by [SAPIUMUsersSyncBegin](#)). Any user with an update time of 0 is removed, as well as his keys, certificates and data objects (e.g., graphical signature image).

This function is relevant only to a Directory Independent environment.

```
SAPIUM int SAPIUMUsersSyncEnd (
    SAPI_UM_SES_HANDLE      Handle,
    SAPI_UM_CONTEXT        SyncContext
);
```

### Parameters

#### Handle

[in] Handle of the SAPIUM session. This handle must have been created using [SAPIUMHandleAcquire](#).

#### SyncContext

[in] Pointer to the Users Sync operation context. This context must have been created using [SAPIUMUsersSyncBegin](#).

### Return Values

If the function succeeds, the return value is SAPI\_OK.

Some possible error codes are listed in the following table.

Value	Meaning
SAPI_ERR_SESSION_IS_NULL	SAPIUM session handle is NULL, was not created using <a href="#">SAPIUMHandleAcquire</a> , or is corrupt.
SAPI_ERR_NULL_VALUE_IS_NOT_ALLOWED	<i>SyncContext</i> should not be NULL.
SAPI_ERR_FAILED_TO_END_SYNC	An error occurred while trying to end the users database synchronization process. Call <a href="#">SAPIUMExtendedLastErrorGet</a> for more information about the exact problem.

### Remarks

The user under whose credentials this operation is performed must have user admin privileges.

**Warning:** Calling the **SAPIUMUsersSyncEnd** function without first calling [SAPIUMUserSync](#) for all the users in the application database, will cause the deletion of all unsynchronized users from the database. Therefore, if the synchronization process is stopped for any reason, do not call **SAPIUMUsersSyncEnd**, but rather restart the whole synchronization process.

### See Also

[SAPIUMHandleAcquire](#), [SAPIUMUsersSyncBegin](#), [SAPIUMUserSync](#), [SAPIUMUsersSyncStop](#)

## SAPIUMUsersSyncStop

The **SAPIUMUsersSyncStop** function stops the users database synchronization process, in such a way that a subsequent call to [SAPIUMUsersSyncEnd](#) will have no effect on the users database. This function is relevant only to a Directory Independent environment.

```
SAPIUM int SAPIUMUsersSyncStop (
    SAPI_UM_SES_HANDLE      Handle,
    SAPI_UM_CONTEXT         SyncContext
);
```

### Parameters

#### Handle

[in] Handle of the SAPIUM session. This handle must have been created using [SAPIUMHandleAcquire](#).

#### SyncContext

[in] Pointer to the Users Sync operation context. This context must have been created using [SAPIUMUsersSyncBegin](#).

### Return Values

If the function succeeds, the return value is SAPI\_OK.

Some possible error codes are listed in the following table.

Value	Meaning
SAPI_ERR_SESSION_IS_NULL	SAPIUM session handle is NULL, was not created using <a href="#">SAPIUMHandleAcquire</a> , or is corrupt.
SAPI_ERR_NULL_VALUE_IS_NOT_ALLOWED	<i>SyncContext</i> should not be NULL.
SAPI_ERR_FAILED_TO_CANCEL_SYNC	An error occurred while trying to stop the users database synchronization process. Call <a href="#">SAPIUMExtendedLastErrorGet</a> for more information about the exact problem.

### Remarks

The user under whose credentials this operation is performed must have user admin privileges.

Stopping the synchronization operation does not cancel the changes made in DocuSign Signature Appliance by the calls to [SAPIUMUsersSyncEnd](#).

### See Also

[SAPIUMHandleAcquire](#), [SAPIUMUsersSyncBegin](#), [SAPIUMUsersSyncEnd](#), [SAPIUMUserSync](#)

## SAPIUMUsersEnumInitEx

The **SAPIUMUsersEnumInitEx** function initializes the continuous operation of retrieving all the users defined in DocuSign Signature Appliance.

```
SAPIUM int SAPIUMUsersEnumInitEx (
    SAPI_UM_SES_HANDLE           Handle,
    SAPI_ENUM_USERS_CONTEXT     UsersEnumContext,
    SAPI_USERS_FILTER_STRUCT    *UsersFilterStruct,
    SAPI_UM_USERS_FILTER_EXT_TYPE UsersFilterExtType,
    void                        *UsersFilterExtValue
);
```

### Parameters

#### Handle

[in] Handle of the SAPIUM session. This handle must have been created using [SAPIUMHandleAcquire](#).

#### UsersEnumContext

[out] Pointer to the Users Enumeration operation context. This context contains internal information about the operation and should be passed to all subsequent calls to [SAPIUMUsersEnumCont](#). The application may not change the value of *UsersEnumContext*; doing so may lead to unexpected behavior.

Note that this type was changed in CoSign version 6.

#### UsersFilterStruct

[in] Pointer to the users filter structure. You can use this structure to indicate which user types (users, computers or groups, as described in [SAPI\\_UM\\_ENUM\\_USER\\_TYPE](#)) to enumerate.

#### UsersFilterExtType

[in] The type of additional filter attribute. Refer to the [SAPI\\_UM\\_USERS\\_FILTER\\_EXT\\_TYPE](#). For example, if the `API_UM_USERS_FILTER_EXT_TYPE_GROUP_TECH_ID` is specified, only users that belong to the specified group will be retrieved.

#### UsersFilterExtValue

[in] The value of the additional filter attribute.

For example, if in the *UsersFilterExtType*, the

`API_UM_USERS_FILTER_EXT_TYPE_GROUP_TECH_ID` is specified, then the provided value should be a pointer to the relevant Group Technical ID.

### Return Values

If the function succeeds, the return value is `SAPI_OK`.

Some possible error codes are listed in the following table.

Value	Meaning
<code>SAPI_ERR_SESSION_IS_NULL</code>	SAPIUM session handle is NULL, was not created using <a href="#">SAPIUMHandleAcquire</a> , or is corrupt.
<code>SAPI_ERR_NULL_VALUE_IS_NOT_ALLOWED</code>	<i>UsersEnumContext</i> should not be NULL.

Value	Meaning
SAPI_ERR_FAILED_TO_ENUM_USERS	An error occurred while trying to initialize the users enumeration operation. Call <a href="#">SAPIUMExtendedLastErrorGet</a> for more information about the exact problem.

**Remarks**

The user under whose credentials this operation is performed must have Users Admin privileges.

**See Also**

[SAPIUMHandleAcquire](#), [SAPIUMUsersEnumCont](#)

## SAPIUMUsersEnumInit

The **SAPIUMUsersEnumInit** function is the older version of the [SAPIUMUsersEnumInitEx](#) function.

```
SAPIUM int SAPIUMUsersEnumInit (
    SAPI_UM_SES_HANDLE      Handle,
    SAPI_ENUM_USERS_CONTEXT UsersEnumContext,
    SAPI_USERS_FILTER_STRUCT *UsersFilterStruct,
);
```

## SAPIUMUsersEnumCont

The **SAPIUMUsersEnumCont** function continues the Users Enumeration operation and returns a pointer to a single user handle. Every call returns a new user handle until the end of the list is reached.

```
SAPIUM int SAPIUMUsersEnumCont (
    SAPI_UM_SES_HANDLE      Handle,
    SAPI_ENUM_USERS_CONTEXT UsersEnumContext,
    SAPI_UM_USR_HANDLE      *UserHandle
);
```

### Parameters

#### Handle

[in] Handle of the SAPIUM session. This handle must have been created using [SAPIUMHandleAcquire](#).

#### UsersEnumContext

[in] Pointer to the Users Enumeration operation context. This context contains internal information about the operation and should be passed to all subsequent calls to **SAPIUMUsersEnumCont**. The application may not change the value of *UsersEnumContext*; doing so may lead to unexpected behavior.

Note that this type was changed in CoSign version 6.

#### UserHandle

[out] Pointer to a handle of a single user. This handle can be used for retrieving more information about the user using the [SAPIUMUserInfoGetEx](#) function.

*UserHandle* should be released by calling the [SAPIUMHandleRelease](#) function, when there is no further use for the handle.

### Return Values

If the function succeeds, the return value is SAPI\_OK.

Some possible error codes are listed in the following table.

Value	Meaning
SAPI_ERR_SESSION_IS_NULL	SAPIUM session handle is NULL, was not created using <a href="#">SAPIUMHandleAcquire</a> , or is corrupt.
SAPI_ERR_NULL_VALUE_IS_NOT_ALLOWED	<i>UsersEnumContext</i> and <i>UserHandle</i> should not be NULL.
SAPI_NO_MORE_USERS	The last user was retrieved, and there are no more users in DocuSign Signature Appliance.
SAPI_FAILED_TO_ALLOCATE_MEMORY	SAPIUM failed to allocate the necessary memory for the new user handle. This might be due to a shortage in system resources.
SAPI_ERR_FAILED_TO_ENUM_USERS	An error occurred while trying to get a single user handle. Call <a href="#">SAPIUMExtendedLastErrorGet</a> for more information about the exact problem.

### Remarks

The user under whose credentials this operation is performed must have Users Admin privileges.

In order to retrieve the whole users list, **SAPIUMUsersEnumCont** should be called until SAPI\_NO\_MORE\_USERS is returned.

***See Also***

[SAPIUMHandleAcquire](#), [SAPIUMUsersEnumInitEx](#), [SAPIUMHandleRelease](#), [SAPIUMUserInfoGetEx](#)

## SAPIUMUserInfoGetEx

The **SAPIUMUserInfoGetEx** function retrieves all the user information stored in DocuSign Signature Appliance for a specific user. The user is specified by a user handle.

```
SAPIUM int SAPIUMUserInfoGet (
    SAPI_UM_SES_HANDLE           Handle,
    SAPI_UM_USR_HANDLE           UserHandle,
    SAPI_LPCWSTR                 UserLoginName,
    long                          *UserLoginNameLen,
    SAPI_LPCWSTR                 UserCN,
    long                          *UserCNLen,
    SAPI_LPCWSTR                 EmailAddress,
    long                          *EmailAddressLen,
    SAPI_ENUM_USER_TYPE         *Kind,
    long                          *RightsMask,
    unsigned long                *UpdateTime,
    SAPI_UM_GUID                 Guid,
    SAPI_UM_ENUM_USER_ENROLLMENT_STATUS *EnrollmentStatus,
    SAPI_UM_ENUM_USER_ENROLLMENT_REASON *EnrollmentReason,
    SAPI_UM_ENUM_USER_LOGIN_STATUS *LoginStatus,
    long                          *Counter1,
    long                          *Counter2,
    long                          *Counter3,
    SAPI_UM_ENUM_USER_CERT_STATUS_TYPE *UserCertStatus,
    SAPI_UM_ENUM_PENDING_REQUEST_STATUS_TYPE *CertRequestStatus
);
```

### Parameters

#### Handle

[in] Handle of the SAPIUM session. This handle must have been created using [SAPIUMHandleAcquire](#).

#### UserHandle

[in] Handle of the user whose information is retrieved. This handle was created by [SAPIUMUsersEnumCont](#).

#### UserLoginName

[out] The name used for logging into DocuSign Signature Appliance. This parameter can be NULL if its value is not required.

#### UserLoginNameLen

[in/out] Pointer to a value that specifies the size allocated for the *UserLoginName* parameter. When the function returns, this value contains the size of the data copied to *UserLoginName*, or if *UserLoginName* is NULL, the size that should be allocated for it. This value can be NULL only if *UserLoginName* is NULL.

#### UserCN

[out] The user's common name. This name is the subject (issued to) of the certificate. This parameter can be NULL if its value is not required.

#### UserCNLen

[in/out] Pointer to a value that specifies the size allocated for the *UserCN* parameter. When the function returns, this value contains the size of the data copied to *UserCN*, or if *UserCN* is NULL, the size that should be allocated for it. This value can be NULL only if *UserCN* is NULL.

### *emailAddress*

[out] The user's email address. This value is part of the subject of the certificate. This parameter can be NULL if its value is not required.

### *emailAddressLen*

[in/out] Pointer to a value that specifies the size allocated for the *emailAddress* parameter. When the function returns, this value contains the size of the data copied to *emailAddress*, or if *emailAddress* is NULL, the size that should be allocated for it. This value can be NULL only if *emailAddress* is NULL.

### *Kind*

[out] Pointer to the user type (user, computer, or group). This parameter can be NULL if its value is not required.

### *RightsMask*

[out] Pointer to a bit mask value of the user rights. See [Appendix I: CoSign Signature Local – User Management Constants](#) for possible values. This parameter can be NULL if its value is not required.

### *UpdateTime*

[out] Pointer to the last time this user was updated. This parameter can be NULL if its value is not required.

### *GUID*

[out] The internal GUID stored for the user.

### *EnrollmentStatus*

[out] The enrollment status of the certificate of the user. See [SAPI UM ENUM USER ENROLLMENT STATUS](#).

### *EnrollmentReason*

[out] The reason for the last certificate enrollment for the user. See [SAPI UM ENUM USER ENROLLMENT REASON](#).

### *LoginStatus*

[out] The logon status of the user. See [SAPI UM ENUM USER LOGIN STATUS](#).

### *Counter1*

[out] The first signature counter of the user.

### *Counter2*

[out] The second signature counter of the user.

### *Counter3*

[out] The third signature counter of the user.

### *UserCertStatus*

[out] The certificate status of the user. See [SAPI UM ENUM USER CERT STATUS TYPE](#).

### *CertRequestStatus*

[out] The certificate request status of the user. See [SAPI UM ENUM PENDING REQUEST STATUS TYPE](#).

### **Return Values**

If the function succeeds, the return value is `SAPI_OK`.

Some possible error codes are listed in the following table.

<b>Value</b>	<b>Meaning</b>
<code>SAPI_ERR_SESSION_IS_NULL</code>	SAPIUM session handle is NULL, was not created using <a href="#">SAPIUMHandleAcquire</a> , or is corrupt.
<code>SAPI_ERR_NULL_VALUE_IS_NOT_ALLOWED</code>	<i>UserHandle</i> and <i>GUID</i> should not be NULL.
<code>SAPI_ERR_FAILED_TO_GET_USER_INFO</code>	An error occurred while trying to get user information. Call <a href="#">SAPIUMExtendedLastErrorGet</a> for more information about the exact problem.

### **Remarks**

The user under whose credentials this operation is performed must have user admin privileges.

### **See Also**

[SAPIUMHandleAcquire](#), [SAPIUMUsersEnumCont](#), [SAPIUMHandleRelease](#), [SAPIUMUserInfoGetEx](#)

## SAPIUMUserInfoGet

The **SAPIUMUserInfoGet** function is the older version of the [SAPIUMUserInfoGetEx](#) function. Note that the extended function [SAPIUMUserInfoGetEx](#) retrieves more information from the user account than **SAPIUMUserInfoGet** does.

```
SAPIUM int SAPIUMUserInfoGet (
    SAPI_UM_SES_HANDLE      Handle,
    SAPI_UM_USR_HANDLE      UserHandle,
    SAPI_LPCWSTR            UserLoginName,
    long                    *UserLoginNameLen,
    SAPI_LPCWSTR            UserCN,
    long                    *UserCNLen,
    SAPI_LPCWSTR            EmailAddress,
    long                    *EmailAddressLen,
    SAPI_ENUM_USER_TYPE     *Kind,
    long                    *RightsMask,
    unsigned long           *UpdateTime,
    SAPI_UM_GUID            Guid
);
```

## SAPIUMCounterReset

The **SAPIUMCounterReset** function resets the signature counters of the given user.

```
SAPIUM int SAPIUMCounterReset (
    SAPI_UM_SES_HANDLE      Handle,
    SAPI_LPCWSTR            UserLoginName,
    SAPI_UM_ENUM_COUNTER_TYPE CounterToReset
);
```

### Parameters

#### Handle

[in] Handle of the SAPIUM session. This handle must have been created using [SAPIUMHandleAcquire](#).

#### UserLoginName

[in] The name used for logging into DocuSign Signature Appliance.

#### CounterToReset

[in] The counter to reset. See [SAPI\\_UM\\_ENUM\\_COUNTER\\_TYPE](#).

### Return Values

If the function succeeds, the return value is SAPI\_OK.

Some possible error codes are listed in the following table.

Value	Meaning
SAPI_ERR_SESSION_IS_NULL	SAPIUM session handle is NULL, was not created using <a href="#">SAPIUMHandleAcquire</a> , or is corrupt.

### Remarks

The user under whose credentials this operation is performed must have user admin privileges.

### See Also

[SAPIUMHandleAcquire](#), [SAPIUMHandleRelease](#), [SAPIUMUserInfoGetEx](#)

## SAPIUMUserAssignGroup

The **SAPIUMUserAssignGroup** function assigns the given user to a group. A user can belong to at most one group.

```
SAPIUM int SAPIUMUserAssignGroup (
    SAPI_UM_SES_HANDLE      Handle,
    SAPI_TECH_ID            UserTechID,
    SAPI_TECH_ID            GroupTechID,
    unsigned long           Flags
);
```

### Parameters

#### Handle

[in] Handle of the SAPIUM session. This handle must have been created using [SAPIUMHandleAcquire](#).

#### UserTechID

[in] The technical identity of the given user.

#### GroupTechID

[in] The technical identity of the given group. 0 indicates that the user does not belong to any group.

#### Flags

[in] Reserved for or future use.

### Return Values

If the function succeeds, the return value is SAPI\_OK.

Some possible error codes are listed in the following table.

Value	Meaning
SAPI_ERR_SESSION_IS_NULL	SAPIUM session handle is NULL, was not created using <a href="#">SAPIUMHandleAcquire</a> , or is corrupt.

### Remarks

The user under whose credentials this operation is performed must have user admin privileges.

When a user is assigned to a group, the user's key and certificate are modified to conform with the group's defined KeySize and OrganizationName values.

### See Also

[SAPIUMHandleAcquire](#), [SAPIUMHandleRelease](#), [SAPIUMUserInfoGetEx](#)

## SAPIUMUserGetByLoginName

The **SAPIUMUserGetByLoginName** function enables you to directly get a user handle and access the user. Previously you could get a user handle only by enumerating all existing users.

```
SAPIUM int SAPIUMUserGetByLoginName (  
    SAPI_UM_SES_HANDLE      Handle,  
    SAPI_LPCWSTR           UserLoginName,  
    SAPI_UM_USR_HANDLE     *UserHandle  
);
```

### Parameters

#### Handle

[in] Handle of the SAPIUM session. This handle must have been created using [SAPIUMHandleAcquire](#).

#### UserLoginName

[in] The name used for logging into DocuSign Signature Appliance.

#### UserHandle

[out] A user handle of the given user name.

### Return Values

If the function succeeds, the return value is SAPI\_OK.

Some possible error codes are listed in the following table.

Value	Meaning
SAPI_ERR_SESSION_IS_NULL	SAPIUM session handle is NULL, was not created using <a href="#">SAPIUMHandleAcquire</a> , or is corrupt.
SAPI_ERR_FAILED_TO_GET_USER	Failed to retrieve the given user.

### Remarks

The user under whose credentials this operation is performed must have user admin privileges.

### See Also

[SAPIUMHandleAcquire](#), [SAPIUMHandleRelease](#), [SAPIUMUserInfoGetEx](#)

## SAPIUMUserTechIDGet

The **SAPIUMUserTechIDGet** gets the technical user identity of a given user. The technical user identity is used in functions such as [SAPIUMUserAssignGroup](#).

```
SAPIUM int SAPIUMUserTechIDGet (
    SAPI_UM_SES_HANDLE      Handle,
    SAPI_UM_USR_HANDLE      UserHandle,
    SAPI_TECH_ID            *UserTechID
);
```

### Parameters

#### Handle

[in] Handle of the SAPIUM session. This handle must have been created using [SAPIUMHandleAcquire](#).

#### UserHandle

[in] A user handle of the given user.

#### UserTechID

[out] The technical identity of the user.

### Return Values

If the function succeeds, the return value is SAPI\_OK.

Some possible error codes are listed in the following table.

Value	Meaning
SAPI_ERR_SESSION_IS_NULL	SAPIUM session handle is NULL, was not created using <a href="#">SAPIUMHandleAcquire</a> , or is corrupt.

### Remarks

The user under whose credentials this operation is performed must have user admin privileges.

### See Also

[SAPIUMHandleAcquire](#), [SAPIUMHandleRelease](#), [SAPIUMUserInfoGetEx](#)

## SAPIUMResetPasswordCounter

The **SAPIUMResetPasswordCounter** resets the password lock counter of a given user.

```
SAPIUM int SAPIUMResetPasswordCounter (  
    SAPI_UM_SES_HANDLE          Handle,  
    SAPI_LPCWSTR                UserLoginName  
);
```

### Parameters

#### Handle

[in] Handle of the SAPIUM session. This handle must have been created using [SAPIUMHandleAcquire](#).

#### UserLoginName

[in] The name used for logging into DocuSign Signature Appliance.

### Return Values

If the function succeeds, the return value is SAPI\_OK.

Some possible error codes are listed in the following table.

Value	Meaning
SAPI_ERR_SESSION_IS_NULL	SAPIUM session handle is NULL, was not created using <a href="#">SAPIUMHandleAcquire</a> , or is corrupt.

### Remarks

The user under whose credentials this operation is performed must have user admin privileges.

### See Also

[SAPIUMHandleAcquire](#), [SAPIUMHandleRelease](#)

## SAPIUMGroupGetByUserGUID

The **SAPIUMGroupGetByUserGUID** retrieves the group record of the group to which the user belongs. If the user does not belong to any group, the technical ID of the group in the GroupRecord output will have a value of 0.

```
SAPIUM int SAPIUMGroupGetByUserGUID (
    SAPI_UM_SES_HANDLE      Handle,
    SAPI_UM_GUID            UserGUID,
    SAPI_UM_GROUP_RECORD    *GroupRecord
);
```

### Parameters

#### Handle

[in] Handle of the SAPIUM session. This handle must have been created using [SAPIUMHandleAcquire](#).

#### UserGUID

[in] The GUID of a given user.

#### GroupRecord

[out] The record of the group to which the user belongs.

### Return Values

If the function succeeds, the return value is SAPI\_OK.

Some possible error codes are listed in the following table.

Value	Meaning
SAPI_ERR_SESSION_IS_NULL	SAPIUM session handle is NULL, was not created using <a href="#">SAPIUMHandleAcquire</a> , or is corrupt.
SAPI_ERR_FAILED_TO_GET_GROUP	Failed to get group information.

### Remarks

The user under whose credentials this operation is performed must have user admin privileges.

### See Also

[SAPIUMHandleAcquire](#), [SAPIUMHandleRelease](#)

## SAPIUMGroupAdd

The **SAPIUMGroupAdd** function adds a group to DocuSign Signature Appliance. This function is relevant only to a Directory Independent environment.

```
SAPIUM int SAPIUMGroupAdd (  
    SAPI_UM_SES_HANDLE           Handle,  
    SAPI_UM_GROUP_RECORD        GroupRecord,  
    SAPI_TECH_ID                GroupTechID,  
);
```

### Parameters

#### Handle

[in] Handle of the SAPIUM session. This handle must have been created using [SAPIUMHandleAcquire](#).

#### GroupRecord

[in] The new group's data. For more information, refer to [SAPI\\_UM\\_GROUP\\_RECORD](#).

#### GroupTechID

[out] The technical identity of the newly created group.

### Return Values

If the function succeeds, the return value is SAPI\_OK.

Some possible error codes are listed in the following table.

Value	Meaning
SAPI_ERR_SESSION_IS_NULL	SAPIUM session handle is NULL, was not created using <a href="#">SAPIUMHandleAcquire</a> , or is corrupt.

### Remarks

The user under whose credentials this operation is performed must have user admin privileges.

If a group with the same name already exists in DocuSign Signature Appliance, an error code is returned.

### See Also

[SAPIUMHandleAcquire](#), [SAPIUMGroupDelete](#), [SAPIUMGroupUpdate](#)

## SAPIUMGroupUpdate

The **SAPIUMGroupUpdate** function updates the group information of a given group. This function is relevant only to a Directory Independent environment. The GroupTechID field inside the given group record indicates which group to update.

```
SAPIUM int SAPIUMGroupUpdate (
    SAPI_UM_SES_HANDLE      Handle,
    SAPI_UM_GROUP_RECORD    GroupRecord
);
```

### **Parameters**

#### *Handle*

[in] Handle of the SAPIUM session. This handle must have been created using [SAPIUMHandleAcquire](#).

#### *GroupRecord*

[in] The new record of the given group (for more information, refer to [SAPI\\_UM\\_GROUP\\_RECORD](#)). The GroupTechID field indicates which group to update.

### **Return Values**

If the function succeeds, the return value is SAPI\_OK.

### **See Also**

[SAPIUMHandleAcquire](#), [SAPIUMGroupAdd](#), [SAPIUMGroupGetByTechID](#), [SAPIUMGroupGetByName](#)

## SAPIUMGroupSetStatusByTechID

The **SAPIUMGroupSetStatusByTechID** function updates the status of the group to either Disabled or Enabled. If the group is Disabled, all users that belong to this group will not be able to login even if their user login state is Enabled. Conversely, if a user is disabled, then even if his/her group is enabled, the user will nevertheless not be able to login.

This function is relevant only to a Directory Independent environment.

```
SAPIUM int SAPIUMGroupSetStatusByTechID (  
    SAPI_UM_SES_HANDLE          Handle,  
    SAPI_TECH_ID                GroupTechID,  
    SAPI_UM__ENUM_GROUP_STATUS_TYPE GroupStatus  
);
```

### **Parameters**

#### *Handle*

[in] Handle of the SAPIUM session. This handle must have been created using [SAPIUMHandleAcquire](#).

#### *GroupTechID*

[in] The technical identity of the group.

#### *GroupStatus*

[in] The new status of the group. For information, refer to [SAPI\\_UM\\_ENUM\\_GROUP\\_STATUS\\_TYPE](#).

### **Return Values**

If the function succeeds, the return value is SAPI\_OK.

### **See Also**

[SAPIUMHandleAcquire](#), [SAPIUMGroupAdd](#)

## SAPIUMGroupDelete

The **SAPIUMGroupDelete** function deletes a group from DocuSign Signature Appliance.

```
SAPIUM int SAPIUMGroupDelete (
    SAPI_UM_SES_HANDLE      Handle,
    SAPI_TECH_ID            GroupTechID,
    unsigned long           UserOP
);
```

### Parameters

#### Handle

[in] Handle of the SAPIUM session. This handle must have been created using [SAPIUMHandleAcquire](#).

#### GroupTechID

[in] The technical identity of the group.

#### UserOP

[in] The parameter value should be 0. For more information, contact ARX.

### Return Values

If the function succeeds, the return value is SAPI\_OK.

Some possible error codes are listed in the following table.

Value	Meaning
SAPI_ERR_SESSION_IS_NULL	SAPIUM session handle is NULL, was not created using <a href="#">SAPIUMHandleAcquire</a> , or is corrupt.

### Remarks

The user under whose credentials this operation is performed must have user admin privileges.

### See Also

[SAPIUMHandleAcquire](#), [SAPIUMGroupAdd](#)

## SAPIUMGroupsEnumInit

The **SAPIUMGroupsEnumInit** function initializes the continuous operation of retrieving all the groups defined in DocuSign Signature Appliance.

```
SAPIUM int SAPIUMGroupsEnumInit (
    SAPI_UM_SES_HANDLE          Handle,
    SAPI_UM_CONTEXT             GroupsEnumContext,
    SAPI_UM_ENUM_GROUPS_ORDER_TYPE GroupsOrderType,
    SAPI_LPCSWSTR               StartValue,
    int                         StartNumberValue,
    int                         OrderDescending
);
```

### Parameters

#### Handle

[in] Handle of the SAPIUM session. This handle must have been created using [SAPIUMHandleAcquire](#).

#### GroupsEnumContext

[in] Pointer to the Groups Enumeration operation context. This context contains internal information about the operation and should be passed to all subsequent calls to [SAPIUMGroupsEnumCont](#). The application may not change the value of *GroupsEnumContext*; doing so may lead to unexpected behavior.

#### GroupsOrderType

[in] Not implemented.

#### StartValue

[in] Not implemented.

#### StartNumberValue

[in] Not implemented.

#### OrderDescending

[in] 0 indicates ascending order, 1 indicates descending order.

### Return Values

If the function succeeds, the return value is SAPI\_OK.

Some possible error codes are listed in the following table.

Value	Meaning
SAPI_ERR_SESSION_IS_NULL	SAPIUM session handle is NULL, was not created using <a href="#">SAPIUMHandleAcquire</a> , or is corrupt.
SAPI_ERR_NULL_VALUE_IS_NOT_ALLOWED	<i>GroupsEnumContext</i> should not be NULL.
SAPI_ERR_FAILED_TO_ENUM_USERS	An error occurred while trying to initialize the groups enumeration operation. Call <a href="#">SAPIUMExtendedLastErrorGet</a> for more information about the exact problem.

**Remarks**

The user under whose credentials this operation is performed must have Users Admin privileges.

**See Also**

[SAPIUMHandleAcquire](#), [SAPIUMGroupsEnumCont](#)

## SAPIUMGroupsEnumCont

The **S SAPIUMGroupsEnumCont** function continues the Groups Enumeration operation and returns a pointer to several group records. Every call returns a bunch of group records until the end of the list is reached.

```
SAPIUM int SAPIUMGroupsEnumCont (
    SAPI_UM_SES_HANDLE      Handle,
    SAPI_UM_CONTEXT         GroupsEnumContext,
    SAPI_UM_GROUP_RECORD    *GroupsList
    unsigned long           *NumOfGroups
);
```

### Parameters

#### Handle

[in] Handle of the SAPIUM session. This handle must have been created using [SAPIUMHandleAcquire](#).

#### GroupsEnumContext

[in] Pointer to the Groups Enumeration operation context. This context contains internal information about the operation and should be passed to all subsequent calls to **SAPIUMGroupsEnumCont**. The application may not change the value of *GroupsEnumContext*; doing so may lead to unexpected behavior.

#### GroupsList

[out] Pointer to an empty list, which will be populated by this function.

#### NumOfGroups

[in/out] Pointer to a value specifying the desired number of group records to return at each call. When the function returns, the value contains the actual number of group records returned.

### Return Values

If the function succeeds, the return value is SAPI\_OK.

Some possible error codes are listed in the following table.

Value	Meaning
SAPI_ERR_SESSION_IS_NULL	SAPIUM session handle is NULL, was not created using <a href="#">SAPIUMHandleAcquire</a> , or is corrupt.
SAPI_ERR_NULL_VALUE_IS_NOT_ALLOWED	<i>GroupsEnumContext</i> , <i>GroupsList</i> and <i>NumOfGroups</i> should not be NULL.
SAPI_NO_MORE_GROUPS	The last group was retrieved, and there are no more groups in DocuSign Signature Appliance.
SAPI_FAILED_TO_ALLOCATE_MEMORY	SAPIUM failed to allocate the necessary memory for the groups context. This might be due to a shortage in system resources.
SAPI_ERR_FAILED_TO_ENUM_GROUPS	An error occurred while trying to get group records. Call <a href="#">SAPIUMExtendedLastErrorGet</a> for more information about the exact problem.

**Remarks**

The user under whose credentials this operation is performed must have Users Admin privileges.

In order to retrieve the records of all existing groups, **SAPIUMGroupsEnumCont** should be called until SAPI\_NO\_MORE\_GROUPS is returned.

**See Also**

[SAPIUMHandleAcquire](#), [SAPIUMGroupsEnumInit](#)

## SAPIUMGroupGetByTechID

The **SAPIUMGroupGetByTechID** retrieves the group record based on a given group's technical ID.

```
SAPIUM int SAPIUMGroupGetByTechID(  
    SAPI_UM_SES_HANDLE      Handle,  
    SAPI_TECH_ID            GroupTechID,  
    SAPI_UM_GROUP_RECORD    *GroupRecord  
);
```

### Parameters

#### Handle

[in] Handle of the SAPIUM session. This handle must have been created using [SAPIUMHandleAcquire](#).

#### GroupTechID

[in] The technical identity of the group.

#### GroupRecord

[out] The group information.

### Return Values

If the function succeeds, the return value is SAPI\_OK.

Some possible error codes are listed in the following table.

Value	Meaning
SAPI_ERR_SESSION_IS_NULL	SAPIUM session handle is NULL, was not created using <a href="#">SAPIUMHandleAcquire</a> , or is corrupt.
SAPI_ERR_FAILED_TO_GET_GROUP	Failed to get group information.

### Remarks

The user under whose credentials this operation is performed must have user admin privileges.

### See Also

[SAPIUMHandleAcquire](#), [SAPIUMGroupAdd](#), [SAPIUMGroupUpdate](#)

## SAPIUMGroupGetByName

The **SAPIUMGroupGetByName** retrieves the group record based on a given group name.

```
SAPIUM int SAPIUMGroupGetByName (
    SAPI_UM_SES_HANDLE      Handle,
    SAPI_LPCTSTR            GroupName,
    SAPI_UM_GROUP_RECORD    *GroupRecord
);
```

### Parameters

#### Handle

[in] Handle of the SAPIUM session. This handle must have been created using [SAPIUMHandleAcquire](#).

#### GroupName

[in] The name of the group.

#### GroupRecord

[out] The group information.

### Return Values

If the function succeeds, the return value is SAPI\_OK.

Some possible error codes are listed in the following table.

Value	Meaning
SAPI_ERR_SESSION_IS_NULL	SAPIUM session handle is NULL, was not created using <a href="#">SAPIUMHandleAcquire</a> , or is corrupt.
SAPI_ERR_FAILED_TO_GET_GROUP	Failed to get group information.

### Remarks

The user under whose credentials this operation is performed must have user admin privileges.

### See Also

[SAPIUMHandleAcquire](#), [SAPIUMGroupAdd](#), [SAPIUMGroupUpdate](#)

## SAPIUMGroupExtDataUpdate

The **SAPIUMGroupExtDataUpdate** updates the extended information of a given group, for example, the maximum allowed number of users in the group.

```
SAPIUM int SAPIUMExtDataUpdate(  
    SAPI_UM_SES_HANDLE           Handle,  
    SAPI_TECH_ID                 GroupTechID,  
    SAPI_UM_ENUM_GROUP_VALUE_NAME ValueType,  
    long                          *IntVal,  
    char                          *StrVal);
```

### Parameters

#### Handle

[in] Handle of the SAPIUM session. This handle must have been created using [SAPIUMHandleAcquire](#).

#### GroupTechID

[in] The technical identity of the group.

#### ValueType

[in] The type of information being updated. For more information refer to the [SAPI\\_UM\\_ENUM\\_GROUP\\_VALUE\\_NAME](#) enumerated type.

#### IntVal

[in] The updated new value if the updated value is an integer type.

#### StrVal

[in] The updated new value if the updated value is a string type.

### Return Values

If the function succeeds, the return value is SAPI\_OK.

Some possible error codes are listed in the following table.

Value	Meaning
SAPI_ERR_SESSION_IS_NULL	SAPIUM session handle is NULL, was not created using <a href="#">SAPIUMHandleAcquire</a> , or is corrupt.

### Remarks

The user performing this operation must have user admin privileges.

### See Also

[SAPIUMHandleAcquire](#), [SAPIUMGroupAdd](#), [SAPIUMGroupUpdate](#), [SAPIUMGroupExtDataGet](#)

## SAPIUMGroupExtDataGet

The **SAPIUMGroupExtDataGet** gets the extended information of a given group.

```
SAPIUM int SAPIUMExtDataUpdate(  
    SAPI_UM_SES_HANDLE           Handle,  
    SAPI_TECH_ID                 GroupTechID,  
    SAPI_UM_ENUM_GROUP_VALUE_NAME ValueType,  
    long                          *IntVal,  
    char                          *StrVal);
```

### Parameters

#### Handle

[in] Handle of the SAPIUM session. This handle must have been created using [SAPIUMHandleAcquire](#).

#### GroupTechID

[in] The technical identity of the group.

#### ValueType

[in] The type of information being retrieved. For more information refer to the [SAPI\\_UM\\_ENUM\\_GROUP\\_VALUE\\_NAME](#) enumerated type.

#### IntVal

[out] The retrieved value if the value is an integer type.

#### StrVal

[out] The retrieved value if the value is a string type.

### Return Values

If the function succeeds, the return value is `SAPI_OK`.

Some possible error codes are listed in the following table.

Value	Meaning
<code>SAPI_ERR_SESSION_IS_NULL</code>	SAPIUM session handle is NULL, was not created using <a href="#">SAPIUMHandleAcquire</a> , or is corrupt.

### Remarks

The user performing this operation must have user admin privileges.

### See Also

[SAPIUMHandleAcquire](#), [SAPIUMGroupAdd](#), [SAPIUMGroupUpdate](#), [SAPIUMGroupExtDataUpdate](#)

## SAPIUMSCPSet

The **SAPIUMSCPSet** function sets the DocuSign Signature Appliance address in the current machine settings. The client on this machine will use this address whenever it needs to connect to DocuSign Signature Appliance.

```
SAPIUM int SAPIUMSCPSet (  
    char *Address  
);
```

### Parameters

*Address*

[in] The DocuSign Signature Appliance address.

### Return Values

If the function succeeds, the return value is `SAPI_OK`.

Some possible error codes are listed in the following table.

Value	Meaning
<code>SAPI_ERR_NULL_VALUE_IS_NOT_ALLOWED</code>	<i>Address</i> should not be NULL.
<code>SAPI_ERR_FAILED_TO_SET_SCP</code>	An error occurred while trying to set the DocuSign Signature Appliance address. Call <a href="#">SAPIUMExtendedLastErrorGet</a> for more information about the exact problem.

### Remarks

Calling this function requires administrative rights on the local machine.

The IP address can also be set by MS group policy configuration. For more information, see the *Distributing the Client Configuration* section in chapter 8 of the *DocuSign Signature Appliance Administrators Guide*.

If the address of DocuSign Signature Appliance has changed, the settings should be changed either by using this function or by using the MS group policy configuration.

This function does not check whether the address is a valid DocuSign Signature Appliance address.



## Signature Local COM Methods for Signing and Verifying

The following sections describe in detail the SAPI COM methods for Signing and Verifying.

### SAPICryptCOM Functions

SAPICryptCOM refers to the CoSign Signature Local COM signing and verifying functions. This chapter lists the SAPICryptCOM interface API functions, and their corresponding SAPICrypt API functions. For a full description of each SAPICryptCOM interface API function, see the description of its corresponding SAPICrypt API function in the SAPICrypt Reference section.

#### Init

This function corresponds to the SAPICrypt [SAPIInit](#) function.

```
Integer Init ()
```

#### HandleAcquire

This function corresponds to the SAPICrypt [SAPIHandleAcquire](#) function.

```
Integer HandleAcquire (
    SESHandle          ref Handle
)
```

#### ConfigurationValueGet

This function corresponds to the SAPICrypt [SAPIConfigurationValueGet](#) function.

```
Integer ConfigurationValueGet(
    SESHandle          Handle,
    SAPI_ENUM_CONF_ID ConfValueID,
    SAPI_ENUM_DATA_TYPE ref ConfValueType,
    VARIANT            ref Value,
    BOOL                SessionValue
)
```

#### ConfigurationValueSet

This function corresponds to the SAPICrypt [SAPIConfigurationValueSet](#) function.

```
Integer ConfigurationValueSet(
    SESHandle          Handle,
    SAPI_ENUM_CONF_ID ConfValueID,
    SAPI_ENUM_DATA_TYPE ConfValueType,
    VARIANT            Value,
    BOOL                SessionValue
)
```

#### ContextRelease

This function corresponds to the SAPICrypt

SAPIContextRelease function.

```
Integer ContextRelease (
    SAPIContext          Context
)
```

## HandleRelease

This function corresponds to the SAPICrypt [SAPIHandleRelease](#) function.

```
Integer HandleRelease (
    SESHandle          Handle
)
```

## Finalize

This function corresponds to the SAPICrypt [SAPIFinalize](#) function.

```
Integer Finalize ()
```

## ExtendedLastErrorGet

This function corresponds to the SAPICrypt [SAPIExtendedLastErrorGet](#) function.

```
Integer ExtendedLastErrorGet ()
```

## LibInfoGet

This function corresponds to the SAPICrypt [SAPILibInfoGet](#) function.

```
Integer LibInfoGet (
    InfoStruct          ref InfoStruct
)
```

## LogonEx2

This function corresponds to the SAPICrypt [SAPILogonEx](#) function.

```
Integer Logon (
    SESHandle          Handle,
    BSTR               UserLoginName,
    BSTR               DomainName,
    SAPIByteArray      Password,
    Integer            Flags,
)
```

## LogonEx

This function corresponds to the SAPICrypt [SAPILogon](#) function. The only difference between **LogonEx** and **Logon** is that in **LogonEx** the password field is a byte array.

```
Integer Logon (
    SESHandle          Handle,
    BSTR               UserLoginName,
    BSTR               DomainName,
    SAPIByteArray      Password
)
```

## Logon

This function corresponds to the SAPICrypt [SAPILogonEx](#) function.

```
Integer Logon (
    SESHandle          Handle,
```

```

    BSTR          UserLoginName,
    BSTR          DomainName,
    BSTR          Password
)

```

## Logoff

This function corresponds to the SAPICrypt [SAPILogoff](#) function.

```

Integer Logoff (
    SESHandle          Handle
)

```

## CredentialChange

This function corresponds to the SAPICrypt [SAPICredentialChange](#) function.

```

Integer CredentialChange (
    SESHandle          Handle,
    BSTR              UserLoginName,
    BSTR              OldPassword,
    BSTR              NewPassword
)

```

## UserActivate

This function corresponds to the SAPICrypt [SAPIUserActivate](#) function.

```

Integer UserAcivate (
    SESHandle          Handle,
    BSTR              UserLoginName,
    BSTR              Password,
    BSTR              NewPassword,
    BSTR              ExtCreds,
    long              Flags
)

```

## CAInfoGetInit

This function corresponds to the SAPICrypt [SAPICAInfoGetInit](#) function.

```

Integer CAInfoGetInit (
    SESHandle          Handle,
    SAPIContext        ref CAInfoContext,
    BSTR              ref CAPublishLocation,
    SAPI_ENUM_CA_INFO_TYPE CAInfoType
)

```

## CAInfoGetCont

This function corresponds to the SAPICrypt [SAPICAInfoGetCont](#) function.

```

Integer CAInfoGetCont (
    SESHandle          Handle,
    SAPIContext        CAInfoContext,
    SAPIByteArray      ref CAInfoBlock,
    BOOL              ref LastBlock
)

```

## CA Cert Install

This function corresponds to the SAPICrypt [SAPICACertInstall](#) function.

```
Integer CACertInstall (
    SESHANDLE          Handle,
    SAPIByteArray      RootCert,
    SAPI_ENUM_STORE_TYPE SystemStoreType
)
```

## GetTokenID

This function corresponds to the [SAPIGetTokenID](#) function.

```
Integer GetTokenID (
    SESHANDLE          Handle,
    BSTR               TokenID
)
```

## GetTokenInfo

This function corresponds to the [SAPIGetTokenInfo](#) function.

```
Integer GetTokenInfo (
    SESHANDLE          Handle,
    BSTR               TokenID,
    long               Flags,
    TokenInfo          *TokenInfo
)
```

## SetTokenID

This function corresponds to the [SAPISetTokenID](#) function.

```
Integer Logon (
    SESHANDLE          Handle,
    BSTR               TokenID
)
```

## TimeGet

This function corresponds to the SAPICrypt [SAPITimeGet](#) function.

```
Integer TimeGet (
    SESHANDLE          Handle,
    SAPIFileTime       ref SystemTime
)
```

## CreateFileHandleByMem

This function corresponds to the SAPICrypt

SAPICreateFileHandleByMem function.

```
Integer CreateFileHandleByMem (
    FileHandle          out FileHandle,
    SAPI_ENUM_FILE_TYPE FileType,
    Integer             Flags,
    SAPI_Byte_Array    FileBuffer
)
```

### CreateFileHandleByName

This function corresponds to the SAPICrypt [SAPICreateFileHandleByName](#) function.

```
Integer CreateFileHandleByName (
    FileHandle          out FileHandle,
    SAPI_ENUM_FILE_TYPE FileType,
    Integer             Flags,
    BSTR                FileUNC
)
```

### GetFileMemData

This function corresponds to the SAPICrypt [SAPIGetFileMemData](#) function.

```
Integer GetFileMemData (
    FileHandle          FileHandle,
    Integer             Flags,
    SAPI_Byte_Array    FileBuffer
)
```

### GetFileHandleType

This function corresponds to the SAPICrypt [SAPIGetFileHandleType](#) function.

```
Integer GetFileHandleType (
    FileHandle          FileHandle,
    SAPI_ENUM_FILE_HANDLE_TYPE out FileHandleType
)
```

### SignatureFieldEnumInitEx

This function corresponds to the SAPICrypt [SAPISignatureFieldEnumInitEx](#) function.

```
Integer SignatureFieldEnumInitEx (
    SESHandle          Handle,
    SAPIContext        ref SigFieldContext,
    SAPI_ENUM_FILE_TYPE FileType,
    BSTR                FileUNC,
    FileHandle          FileHandle,
    Integer             Flags,
    Integer             ref SignatureFieldsNum
)
```

### SignatureFieldEnumInit

This function corresponds to the SAPICrypt **SAPISignatureFieldEnumInit** function, which is the older version of the [SAPISignatureFieldEnumInitEx](#) function.

```
Integer SignatureFieldEnumInit (
    SESHandle          Handle,
    SAPIContext        ref SigFieldContext,
    SAPI_ENUM_FILE_TYPE FileType,
    BSTR                FileUNC,
)
```

```

Integer          Flags,
Integer          ref SignatureFieldsNum
)

```

### SignatureFieldEnumCont

This function corresponds to the SAPICrypt [SAPISignatureFieldEnumCont](#) function.

```

Integer SignatureFieldEnumCont (
    SESHandle          Handle,
    SAPIContext        SigFieldContext,
    sigFieldHandle     ref SignatureFieldHandle
)

```

### SignatureFieldLocatorEnumInit

This function corresponds to the SAPICrypt [SAPISignatureFieldLocatorEnumInit](#) function.

```

Integer SignatureFieldEnumInit (
    SESHandle          Handle,
    SAPIContext        SigFieldLocatorContext,
    FileHandle         FileHandle,
    BSTR               OpeningPatern,
    BSTR               ClosingPatern,
    Integer            Flags,
    Integer            out SignatureFieldsLocatedNum
)

```

### SignatureFieldLocatorEnumCont

This function corresponds to the SAPICrypt [SAPISignatureFieldLocatorEnumCont](#) function.

```

Integer SignatureFieldEnumCont (
    SESHandle          Handle,
    SAPIContext        SigFieldLocatorContext,
    SigFieldSettings   SigFieldSettings,
    BSTR               out EncodedMessage
)

```

### SignatureFieldInfoGet

This function corresponds to the SAPICrypt [SAPISignatureFieldInfoGet](#) function.

```

Integer SignatureFieldInfoGet (
    SESHandle          Handle,
    SigFieldHandle     SignatureFieldHandle,
    SigFieldSettings   ref SignatureFieldSettingStruct,
    SigFieldInfo       ref SignedFieldInfoStruct
)

```

### SignatureFieldCreateEx

This function corresponds to the SAPICrypt [SAPISignatureFieldCreateEx](#) function.

```

Integer SignatureFieldCreateEx (
    SESHandle          Handle,
    SAPI_ENUM_FILE_TYPE FileType,
    BSTR               FileUNC,
    FileHandle         FileHandle,
    SigFieldSettings   NewSignatureFieldStruct,
    Integer            Flags,
    SigFieldHandle     ref SignatureFieldHandle
)

```

## SignatureFieldCreate

This function corresponds to the SAPICrypt [SAPISignatureFieldCreate](#) function, which is the older version of the [SAPISignatureFieldCreateEx](#) function.

```
Integer SignatureFieldCreate (
    SESHandle          Handle,
    SAPI_ENUM_FILE_TYPE FileType,
    BSTR               FileUNC,
    SigFieldSettings   NewSignatureFieldStruct,
    Integer            Flags,
    SigFieldHandle     ref SignatureFieldHandle
)
```

## SignatureFieldSignEx2

This function corresponds to the SAPICrypt [SAPISignatureFieldSignEx](#) function. The difference between **SignaturefieldSignEx** and **SignaturefieldSignEx2** is that in **SignaturefieldSignEx2** the credentials are passed as a byte array.

```
Integer SignatureFieldSignEx (
    SESHandle          Handle,
    SigFieldHandle     SignatureFieldHandle,
    Integer            Flags,
    SAPIByteArray      Credential
)
```

## SignatureFieldSignEx

This function corresponds to the SAPICrypt [SAPISignatureFieldSignEx](#) function.

```
Integer SignatureFieldSignEx (
    SESHandle          Handle,
    SigFieldHandle     SignatureFieldHandle,
    Integer            Flags,
    BSTR               Credential
)
```

## SignatureFieldSign

This function corresponds to the SAPICrypt [SAPISignatureFieldSign](#) function, which is the older version of the [SAPISignatureFieldSignEx](#) function.

```
Integer SignatureFieldSign (
    SESHandle          Handle,
    SigFieldHandle     SignatureFieldHandle,
    Integer            Flags
)
```

## SignatureFieldCreateSignEx2

This function corresponds to the SAPICrypt

SAPISignatureFieldCreateSignEx function.

```
Integer SignatureFieldCreateSignEx2 (  
    SESHandle          Handle,  
    SAPI_ENUM_FILE_TYPE FileType,  
    BSTR               FileUNC,  
    FileHandle         FileHandle,  
    SigFieldSettings   NewSignatureFieldStruct,  
    Integer             Flags,  
    SAPIByteArray       Credential  
)
```

### **SignatureFieldCreateSignEx**

This function corresponds to the SAPICrypt [SAPISignatureFieldCreateSign](#) function, which is the older version of the

SAPISignatureFieldCreateSignEx function.

The difference between **SignatureFieldCreateSign** and **SignatureFieldCreateSignEx** is that in **SignatureFieldCreateSignEx** the credentials are passed as a byte array.

```
Integer SignatureFieldCreateSign (  
    SESHandle          Handle,  
    SAPI_ENUM_FILE_TYPE FileType,  
    BSTR              FileUNC,  
    SigFieldSettings  NewSignatureFieldStruct,  
    Integer           Flags,  
    SAPIByteArray     Credential  
)
```

### **SignatureFieldCreateSign**

This function corresponds to the SAPICrypt [SAPISignatureFieldCreateSign](#) function, which is the older version of the

SAPISignatureFieldCreateSignEx function.

```
Integer SignatureFieldCreateSign (
    SESHandle          Handle,
    SAPI_ENUM_FILE_TYPE FileType,
    BSTR               FileUNC,
    SigFieldSettings  NewSignatureFieldStruct,
    Integer            Flags,
    BSTR               Credential
)
```

## SignatureFieldVerify

This function corresponds to the SAPICrypt [SAPISignatureFieldVerify](#) function.

```
Integer SignatureFieldVerify (
    SESHandle          Handle,
    SigFieldHandle     SignatureFieldHandle,
    CertStatus         ref CertificateStatus,
    Integer            Flags
)
```

## FileIsSignedEx

This function corresponds to the SAPICrypt [SAPIFileIsSignedEx](#) function.

```
Integer FileIsSignedEx (
    SESHandle          Handle,
    SAPI_ENUM_FILE_TYPE FileType,
    BSTR               FileUNC,
    FileHandle         FileHandle,
    Integer            Flags,
    BOOL               ref IsSigned,
)
```

## FileIsSigned

This function corresponds to the SAPICrypt [SAPIFileIsSigned](#) function, which is the older version of the [SAPIFileIsSignedEx](#) function.

```
Integer FileIsSigned (
    SESHandle          Handle,
    SAPI_ENUM_FILE_TYPE FileType,
    BSTR               FileUNC,
    Integer            Flags,
    BOOL               ref IsSigned,
)
```

## SignatureFieldClear

This function corresponds to the SAPICrypt [SAPISignatureFieldClear](#) function.

```
Integer SignatureFieldClear (
    SESHandle          Handle,
    SigFieldHandle     SignatureFieldHandle,
    Integer            Flags
)
```

## SignatureFieldRemove

This function corresponds to the SAPICrypt [SAPISignatureFieldRemove](#) function.

```
Integer SignatureFieldRemove (
    SESHandle          Handle,
    SigFieldHandle     SignatureFieldHandle,
    Integer            Flags
)
```

## SignatureFieldDetailsGUIGet

This function corresponds to the SAPICrypt [SAPISignatureFieldDetailsGUIGet](#) function.

```
Integer SignatureFieldRemove (
    SESHandle          Handle,
    SigFieldHandle     SignatureFieldHandle
)
```

## CertificatesEnumInit

This function corresponds to the SAPICrypt [SAPICertificatesEnumInit](#) function.

```
Integer CertificatesEnumInit (
    SESHandle          Handle,
    SAPIContext        ref CertContext,
    Integer            Flags
)
```

## CertificatesEnumCont

This function corresponds to the SAPICrypt [SAPICertificatesEnumCont](#) function.

```
Integer CertificatesEnumCont (
    SESHandle          Handle,
    SAPIContext        CertContext,
    ICertHandle        ref CertHandle
)
```

## CertificateGetFieldByHandle

This function corresponds to the SAPICrypt [SAPICertificateGetFieldByHandle](#) function.

```
Integer CertificateGetFieldByHandle (
    SESHandle          Handle,
    ICertHandle        CertHandle,
    SAPI_ENUM_CERT_FIELD FieldID,
    VARIANT            ref Value,
    SAPI_ENUM_DATA_TYPE ref FieldType
)
```

## CertificateGetFieldByBlob

This function corresponds to the SAPICrypt [SAPICertificateGetFieldByBlob](#) function.

```
Integer CertificateGetFieldByBlob (
    SESHandle          Handle,
    SAPIByteArray      EncodedCertificate,
    SAPI_ENUM_CERT_FIELD FieldID,
    VARIANT            ref Value,
    SAPI_ENUM_DATA_TYPE__RPC_FAR ref FieldType
)
```

## PKCS7BlobGetValue

This function corresponds to the SAPICrypt [SAPIPKCS7BlobGetValue](#) function.

```
Integer PKCS7BlobGetValue (
    SESHandle          Handle,
    SAPIByteArray      PKCS7Blob,
    SAPI_ENUM_PKCS7_FIELD FieldID,
    VARIANT            ref Value,
    SAPI_ENUM_DATA_TYPE ref FieldType
)
```

## CertificateGUISelect

This function corresponds to the SAPICrypt [SAPICertificateGUISelect](#) function.

```
Integer CertificateGUISelect (
    SESHandle          Handle,
    Integer            Flags,
    ICertHandle        ref CertHandle
)
```

## CertificateSetDefault

This function corresponds to the SAPICrypt [SAPICertificateSetDefault](#) function.

```
Integer CertificateSetDefault (
    SESHandle          Handle,
    ICertHandle        CertHandle
)
```

## CertificateGetDefault

This function corresponds to the SAPICrypt [SAPICertificateGetDefault](#) function.

```
Integer CertificateGetDefault (
    SESHandle          Handle,
    ICertHandle        ref CertHandle
)
```

## GraphicSigImageEnumInit

This function corresponds to the SAPICrypt [SAPIGraphicSigImageEnumInit](#) function.

```
Integer GraphicSigImageEnumInit (
    SESHandle          Handle,
    SAPIContext        ref GraphicImageContext,
    Integer            FormatsPermitted,
    Integer            Flags
)
```

## GraphicSigImageEnumCont

This function corresponds to the SAPICrypt [SAPIGraphicSigImageEnumCont](#) function.

```
Integer GraphicSigImageEnumCont (
    SESHandle          Handle,
    SAPIContext        GrImageContext,
    GraphicImageHandle ref GraphicImageHandle
)
```

## GraphicSigImageGUISelect

This function corresponds to the SAPICrypt [SAPIGraphicSigImageGUISelect](#) function.

```
Integer GraphicSigImageGUISelect (
    SESHandle          Handle,
    Integer            FormatsPermitted,
    Integer            Flags,
    SAPI_ENUM_GR_IMAGE_SELECT_MODE selectMode,
    GraphicImageHandle ref GraphicImageHandle
)
```

## SigningCeremonyGUI

This function corresponds to the SAPICrypt [SAPISigningCeremonyGUI](#) function.

```
Integer SigningCeremonyGUI (
    SESHandle          Handle,
    Integer            Flags,
    Integer            DisplayComponentsMask,
    Integer            FormatsPermitted,
    BSTR               InstructToSigner,
    SAPI_ENUM_CEREMONY_COMPONENT_SHOW_MODE ImageSelection,
    SAPI_ENUM_CEREMONY_COMPONENT_SHOW_MODE CertSelection,
    SAPI_ENUM_CEREMONY_COMPONENT_SHOW_MODE ReasonSelection,
    SAPI_ENUM_CEREMONY_COMPONENT_SHOW_MODE TitleSelection,
    SAPI_ENUM_GR_IMAGE_SELECT_MODE         selectMode
)
```

## GraphicSigImageSetDefault

This function corresponds to the SAPICrypt [SAPIGraphicSigImageSetDefault](#) function, which is the older version of the [SAPIGraphicSigImageSetDefaultEx](#) function.

```
Integer GraphicSigImageSetDefault (
    SESHandle          Handle,
    GraphicImageHandle GraphicImageHandle
)
```

## GraphicSigImageSetDefaultEx

This function corresponds to the SAPICrypt [SAPIGraphicSigImageSetDefaultEx](#) function.

```
Integer GraphicSigImageSetDefaultEx (
    SESHandle          Handle,
    GraphicImageHandle GraphicImageHandle,
    Integer            Flags
)
```

## GraphicSigImageGetDefault

This function corresponds to the SAPICrypt [SAPIGraphicSigImageGetDefault](#) function, which is the older version of the [SAPIGraphicSigImageGetDefaultEx](#) function.

```
Integer GraphicSigImageGetDefault (
    SESHandle          Handle,
    GraphicImageHandle ref GraphicImageHandle
)
```

## GraphicSigImageGetDefaultEx

This function corresponds to the SAPICrypt [SAPIGraphicSigImageGetDefaultEx](#) function.

```
Integer GraphicSigImageGetDefaultEx (  
    SESHandle          Handle,  
    GraphicImageHandle ref GraphicImageHandle,  
    Integer            Flags  
)
```

## GraphicSigImageInfoGet

This function corresponds to the SAPICrypt [SAPIGraphicSigImageInfoGet](#) function.

```
Integer GraphicSigImageInfoGet (  
    SESHandle          Handle,  
    GraphicImageHandle GraphicImageHandle,  
    GraphicImageInfo  ref GraphicImageInfo,  
    SAPI_ENUM_GRAPHIC_IMAGE_FORMAT Convert,  
    Integer            Flags  
)
```

## GraphicSigImageInfoCreate

This function corresponds to the SAPICrypt [SAPIGraphicSigImageInfoCreate](#) function.

```
Integer GraphicSigImageInfoCreate (  
    SESHandle          Handle,  
    GraphicImageHandle ref GraphicImageHandle,  
    GraphicImageInfo  ref GraphicImageInfo  
)
```

## GraphicSigInitialsInfoCreate

This function corresponds to the SAPICrypt [SAPIGraphicSigInitialsInfoCreate](#) function.

```
Integer GraphicSigImageInfoCreate (  
    SESHandle          Handle,  
    GraphicImageHandle ref GraphicImageHandle,  
    GraphicImageInfo  ref GraphicImageInfo  
)
```

## GraphicSigLogoInfoCreate

This function corresponds to the SAPICrypt [SAPIGraphicSigLogoInfoCreate](#) function.

```
Integer GraphicSigLogoInfoCreate (  
    SESHandle          Handle,  
    GraphicImageHandle ref GraphicImageHandle,  
    GraphicImageInfo  ref GraphicImageInfo  
)
```

## GraphicSigImageInfoDelete

This function corresponds to the SAPICrypt [SAPIGraphicSigImageInfoDelete](#) function.

```
Integer GraphicSigImageInfoDelete (  
    SESHandle          Handle,  
    GraphicImageHandle GraphicImageHandle  
)
```

## GraphicSigImageInfoUpdate

This function corresponds to the SAPICrypt [SAPIGraphicSigImageInfoUpdate](#) function.

```
Integer GraphicSigImageInfoUpdate (
    SESShandle          Handle,
    GraphicImageHandle  GraphicImageHandle,
    GraphicImageInfo    GraphicImageInfo,
    Integer             Flags
)
```

## GraphicSigInitialsInfoUpdate

This function corresponds to the SAPICrypt [SAPIGraphicSigInitialsInfoUpdate](#) function.

```
Integer GraphicSigInitialsInfoUpdate (
    SESShandle          Handle,
    GraphicImageHandle  GraphicImageHandle,
    GraphicImageInfo    GraphicImageInfo,
    Integer             Flags
)
```

## GraphicSigLogoInfoUpdate

This function corresponds to the SAPICrypt [SAPIGraphicSigLogoInfoUpdate](#) function.

```
Integer GraphicSigLogoInfoUpdate (
    SESShandle          Handle,
    GraphicImageHandle  GraphicImageHandle,
    GraphicImageInfo    GraphicImageInfo,
    Integer             Flags
)
```

## BufferSignEx2

This function corresponds to the SAPICrypt **SAPIBufferSignEx** function.

The difference between **BufferSignEx** and **BufferSignEx2** is that in **BufferSignEx2** the credentials are passed as a byte array.

```
Integer BufferSignEx (
    SESShandle          Handle,
    SAPIByteArray       Buffer,
    SAPIByteArray       SignedData,
    Integer             Flags,
    SAPIByteArray       Credential
)
```

## BufferSignEx

This function corresponds to the SAPICrypt [SAPIBufferSignEx](#) function.

```
Integer BufferSignEx (
    SESShandle          Handle,
    SAPIByteArray       Buffer,
    SAPIByteArray       SignedData,
    Integer             Flags,
    BSTR                Credential
)
```

## BufferSign

This function corresponds to the SAPICrypt [SAPIBufferSign](#) function, which is the older version of the [SAPIBufferSignEx](#) function.

```
Integer BufferSign (
    SESHandle          Handle,
    SAPIByteArray      Buffer,
    SAPIByteArray      SignedData,
    Integer             Flags
)
```

## BufferSignInit

This function corresponds to the SAPICrypt [SAPIBufferSignInit](#) function.

```
Integer BufferSignInit (
    SESHandle          Handle,
    SAPIContext        ref SignContext,
    Integer             Flags
)
```

## BufferSignCont

This function corresponds to the SAPICrypt [SAPIBufferSignCont](#) function.

```
Integer BufferSignCont (
    SESHandle          Handle,
    SAPIContext        SignContext,
    SAPIByteArray      Buffer
)
```

## BufferSignEndEx2

This function corresponds to the SAPICrypt [SAPIBufferSignEndEx](#) function.

The difference between **BufferSignEndEx** and **BufferSignEndEx2** is that in **BufferSignEndEx2** the credentials are passed as a byte array.

```
Integer BufferSignEndEx (
    SESHandle          Handle,
    SAPIContext        SignContext,
    SAPIByteArray      SignedData,
    SAPIByteArray      Credential
)
```

## BufferSignEndEx

This function corresponds to the SAPICrypt [SAPIBufferSignEndEx](#) function.

```
Integer BufferSignEndEx (
    SESHandle          Handle,
    SAPIContext        SignContext,
    SAPIByteArray      SignedData,
    BSTR               Credential
)
```

## BufferSignEnd

This function corresponds to the SAPICrypt [SAPIBufferSignEnd](#) function, which is the older version of the [SAPIBufferSignEndEx](#) function.

```
Integer BufferSignEnd (
    SESHandle          Handle,
```

```

SAPIContext          SignContext,
SAPIByteArray        ref SignedData
)

```

### SigningContextPKCS7BlobLenGet

This function corresponds to the SAPICrypt [SAPISigningContextPKCS7BlobLenGet](#) function.

```

Integer SigningContextPKCS7BlobLenGet (
    SESHandle          Handle,
    SAPIContext        SignContext,
    Integer            ref SignatureLen
)

```

### BufferVerifySignature

This function corresponds to the SAPICrypt [SAPIBufferVerifySignature](#) function.

```

Integer BufferVerifySignature (
    SESHandle          Handle,
    SAPIByteArray      Buffer,
    SAPIByteArray      SignedData,
    SAPIFileTime       ref SignatureTime,
    CertStatus         ref CertificateStatus,
    Integer            Flags
)

```

### BufferVerifySignatureInit

This function corresponds to the SAPICrypt [SAPIBufferVerifySignatureInit](#) function.

```

Integer BufferVerifySignatureInit (
    SESHandle          Handle,
    SAPIContext        ref VerifyContext,
    SAPIByteArray      SignedData,
    Integer            Flags
)

```

### BufferVerifySignatureCont

This function corresponds to the SAPICrypt [SAPIBufferVerifySignatureCont](#) function.

```

Integer BufferVerifySignatureCont (
    SESHandle          Handle,
    SAPIContext        VerifyContext,
    SAPIByteArray      Buffer
)

```

### BufferVerifySignatureEnd

This function corresponds to the SAPICrypt [SAPIBufferVerifySignatureEnd](#) function.

```

Integer BufferVerifySignatureEnd (
    SESHandle          Handle,
    SAPIContext        VerifyContext,
    SAPIFileTime       ref SignatureTime,
    CertStatus         ref CertificateStatus
)

```

### LoginTicketCheckInit, LoginTicketCheckCont, and LoginTicketCheckEnd

Contact ARX for more information.



## SAPICOM Samples

The following samples illustrate the usage of the SAPICOM interface. These samples are located on the SAPI SDK CD in the `samples\SAPICOM` directory.

### Sample 1 – Buffer Signing Code Using VB.NET

The following sample is code that illustrates signing a simple buffer in a VB .NET project.

```
Rem The following function is called as part of a signature operation.
Rem The digital signature in a base64 encoding will be directed to a special field.

Private Sub Sign_Click(ByVal sender As System.Object, ByVal e As System.EventArgs)
Handles Sign.Click

    Dim SAPI As New SAPICrypt
    Dim MyHandle As New SESHANDLE
    Dim MyInputData As New SAPIByteArray
    Dim MySignature As New SAPIByteArray
    Dim rc As Int32
    Dim Enc As New System.Text.ASCIIEncoding
    Rem Convert the string input data to binary information.
    MyInputData.FromArray(Enc.GetBytes(Data.Text))
    Rem Initialize SAPI.
    rc = SAPI.Init()
    If rc <> 0 Then
        MsgBox("Init failed: " + CStr(Hex(rc)))
        Exit Sub
    End If
    Rem Acquire the session handle.
    rc = SAPI.HandleAcquire(MyHandle)
    If rc <> 0 Then
        MsgBox("HandleAcquire failed: " + CStr(Hex(rc)))
        SAPI.Finalize()
        Exit Sub
    End If
    Rem If necessary, login (depending on whether the User ID has a value.
    If UserID.Text <> "" Then
        rc = SAPI.Logon(MyHandle, UserID.Text, Domain.Text, Password.Text)
        If rc <> 0 Then
            MsgBox("Login failed: " + CStr(Hex(rc)))
            SAPI.HandleRelease(MyHandle)
            SAPI.Finalize()
            Exit Sub
        End If
    End If
    Rem Sign the given buffer.
    rc = SAPI.BufferSign(MyHandle, MyInputData, MySignature, 0)
    If rc <> 0 Then
        MsgBox("BufferSign failed: " + CStr(Hex(rc)))
        SAPI.HandleRelease(MyHandle)
        SAPI.Finalize()
        Exit Sub
    End If
    Rem Convert the signature to a Base64 string.
    Signature.Text = Convert.ToBase64String(MySignature.ToArray())
    Signature.Refresh()
    Rem Logoff.
    If UserID.Text <> "" Then
        SAPI.Logoff(MyHandle)
    End If
    Rem Release resources.
```

```

    SAPI.HandleRelease(MyHandle)
    SAPI.Finalize()
End Sub

```

This function is called from a GUI application that enables an end user to sign a given data buffer.

This function uses either the default user, or a given user's credentials, depending on the value of the UserID field.

## Sample 2 – Buffer Verification Code Using VB .NET

The following sample is code that illustrates verifying a simple buffer in a VB .NET project.

```

Rem This function will verify the existing digital signature.
Rem A success message box will be displayed upon successful verification.
Private Sub Verify_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Verify.Click
    Dim SAPI As New SAPICrypt
    Dim MyHandle As New SESHandle
    Dim MyInputData As New SAPIByteArray
    Dim MySignature As New SAPIByteArray
    Dim rc As Int32
    Dim Enc As New System.Text.ASCIIEncoding
    Dim SigTime As New SAPIFileTime
    Dim CertStatus As New CertStatus
    Rem Get the original data.
    MyInputData.FromArray(Enc.GetBytes(Data.Text))
    Rem Convert the signature from base64 format to binary format.
    MySignature.FromArray(Convert.FromBase64String(Signature.Text))
    Rem Initialize SAPI.
    rc = SAPI.Init()
    If rc <> 0 Then
        MsgBox("Init failed: " + CStr(Hex(rc)))
        Exit Sub
    End If
    Rem Acquire the session handle.
    rc = SAPI.HandleAcquire(MyHandle)
    If rc <> 0 Then
        MsgBox("HandleAcquire failed: " + CStr(Hex(rc)))
        SAPI.Finalize()
        Exit Sub
    End If
    Rem If necessary, login depending on the UserID field.
    If UserID.Text <> "" Then
        rc = SAPI.Logon(MyHandle, UserID.Text, Domain.Text, Password.Text)
        If rc <> 0 Then
            MsgBox("Login failed: " + CStr(Hex(rc)))
            SAPI.HandleRelease(MyHandle)
            SAPI.Finalize()
            Exit Sub
        End If
    End If
    Rem Verify the digital signature.
    rc = SAPI.BufferVerifySignature(MyHandle, MyInputData, MySignature,
        SigTime, CertStatus, 0)
    If rc <> 0 Then
        MsgBox("BufferVerifySignature failed: " + CStr(Hex(rc)))
        SAPI.HandleRelease(MyHandle)
        SAPI.Finalize()
        Exit Sub
    Else
        MsgBox("Success!")
    End If
End Sub

```

```
Rem Logoff.  
If UserID.Text <> "" Then  
    SAPI.Logoff(MyHandle)  
End If  
Rem Release resources.  
SAPI.HandleRelease(MyHandle)  
SAPI.Finalize()  
End Sub
```

This function is called from a GUI application that enables an end user to verify a signature of a given data buffer.



## Signature Local COM Methods for User Management

The following sections describe in detail the SAPIUM COM methods for User Management.

### SAPIUMCOM Functions

SAPIUMCOM refers to the CoSign Signature Local COM user management functions. This chapter lists the SAPIUMCOM functions, and their corresponding SAPIUM API functions. For a detailed description of each SAPIUMCOM interface API function, see the description of its corresponding SAPIUM API function in the SAPIUM Reference section.

#### Init

This function corresponds to the SAPIUM [SAPIUMInit](#) function.

```
Integer Init ()
```

#### HandleAcquire

This function corresponds to the SAPIUM [SAPIUMHandleAcquire](#) function.

```
Integer HandleAcquire (  
    UMSESHandle          ref Handle  
)
```

#### HandleRelease

This function corresponds to the SAPIUM [SAPIUMHandleRelease](#) function.

```
Integer HandleRelease (  
    UMSESHandle          Handle  
)
```

#### Finalize

This function corresponds to the SAPIUM [SAPIUMFinalize](#) function.

```
Integer Finalize ()
```

#### ExtendedLastErrorGet

This function corresponds to the SAPIUM [SAPIUMExtendedLastErrorGet](#) function.

```
Integer ExtendedLastErrorGet ()
```

#### LibInfoGet

This function corresponds to the SAPIUM [SAPIUMLibInfoGet](#) function.

```
Integer LibInfoGet (  
    InfoStruct          ref InfoStruct  
)
```

#### Logon

This function corresponds to the SAPIUM [SAPIUMLogon](#) function.

```
Integer Logon (  
    UMSESHandle          Handle,  
    BSTR                 UserLoginName,  
    BSTR                 DomainName,
```

```
BSTR Password
)
```

## LogonEx

This function corresponds to the SAPIUM [SAPIUMLogonEx](#) function.

```
Integer LogonEx (
    UMSESHandle Handle,
    BSTR UserLoginName,
    BSTR DomainName,
    SAPIUMByteArray Password,
    Integer Flags
)
```

## Logoff

This function corresponds to the SAPIUM [SAPIUMLogoff](#) function.

```
Integer Logoff (
    UMSESHandle Handle
)
```

## GetTokenID

This function corresponds to the

SAPIUMGetTokenID function.

```
Integer GetTokenID (
    UMSESHandle          Handle,
    BSTR                 TokenID
)
```

## SetTokenID

This function corresponds to the [SAPIUMSetTokenID](#) function.

```
Integer Logon (
    UMSESHandle          Handle,
    BSTR                 TokenID
)
```

## UserAddEx1

This function corresponds to the SAPIUM [SAPIUMUserAddEx1](#) function.

```
Integer UserAddEx1 (
    UMSESHandle          Handle,
    BSTR                 UserLoginName,
    BSTR                 UserCN,
    BSTR                 EmailAddress,
    BSTR                 Password,
    long                 GroupTechID,
    Integer              Flags
)
```

## UserAdd

This function corresponds to the SAPIUM [SAPIUMUserAdd](#) function, which is the older version of the [SAPIUMUserAddEx1](#) function, dating to before DocuSign Signature Appliance groups were introduced.

```
Integer UserAdd (
    UMSESHandle          Handle,
    BSTR                 UserLoginName,
    BSTR                 UserCN,
    BSTR                 EmailAddress,
    BSTR                 Password,
    Integer              Flags
)
```

## UserUpdate

This function corresponds to the SAPIUM [SAPIUMUserUpdate](#) function.

```
Integer UserUpdate (
    UMSESHandle          Handle,
    BSTR                 UserLoginName,
    BSTR                 UserCN,
    BSTR                 EmailAddress,
    Integer              Flags
)
```

## CredentialSet

This function corresponds to the SAPIUM [SAPIUMCredentialSet](#) function.

```
Integer CredentialSet (
    UMSESHandle          Handle,
    BSTR                 UserLoginName,
```

```
BSTR Password
)
```

## UserSetLogonState

This function corresponds to the SAPIUM [SAPIUMUserSetLogonState](#) function.

```
Integer UserSetLogonState (
    UMSESHandle          Handle,
    BSTR                 UserLoginName,
    SAPI_UM_ENUM_USER_LOGIN_STATUS User  UserLogonStatus
);
```

## CounterReset

This function corresponds to the SAPIUM [SAPIUMCounterReset](#) function.

```
Integer CounterReset (
    UMSESHandle          Handle,
    BSTR                 UserLoginName,
    SAPI_UM_ENUM_COUNTER_TYPE CounterToReset
)
```

## UserAssignGroup

This function corresponds to the SAPIUM [SAPIUMUserAssignGroup](#) function.

```
Integer UserAssignGroup (
    UMSESHandle          Handle,
    long                 UserTechID,
    long                 GroupTechID,
    Integer              Flags
)
```

## UserDelete

This function corresponds to the SAPIUM [SAPIUMUserDelete](#) function.

```
Integer UserDelete (
    UMSESHandle          Handle,
    BSTR                 UserLoginName
)
```

## UsersSyncBegin

This function corresponds to the SAPIUM [SAPIUMUsersSyncBegin](#) function.

```
IntegerUsersSyncBegin (
    UMSESHandle          Handle,
    UMContext            ref SyncContext
)
```

## UserSync

This function corresponds to the SAPIUM [SAPIUMUserSync](#) function.

```
Integer UserSync (
    UMSESHandle          Handle,
    UMContext            SyncContext,
    BSTR                 UserLoginName,
    BSTR                 UserCN,
    BSTR                 EmailAddress,
    BSTR                 Password,
)
```

```
Integer          Flags
)
```

## UsersSyncEnd

This function corresponds to the SAPIUM [SAPIUMUsersSyncEnd](#) function.

```
Integer UsersSyncEnd (
    UMSESHandle          Handle,
    UMContext            SyncContext
)
```

## UsersSyncStop

This function corresponds to the SAPIUM [SAPIUMUsersSyncStop](#) function.

```
Integer UsersSyncStop (
    UMSESHandle          Handle,
    UMContext            SyncContext
)
```

## UsersEnumInit

This function corresponds to the SAPIUM [SAPIUMUsersEnumInit](#) function.

```
Integer UsersEnumInit (
    UMSESHandle          Handle,
    UMContext            UsersEnumContext,
    UsersFilter          UsersFilterStruct
)
```

## UsersEnumCont

This function corresponds to the SAPIUM [SAPIUMUsersEnumCont](#) function.

```
Integer UsersEnumCont (
    UMSESHandle          Handle,
    UMContext            UsersEnumContext,
    UserHandle           ref UserHandle
)
```

## UserInfoGet

This function corresponds to the SAPIUM [SAPIUMUserInfoGet](#) function, which is the older version of the [SAPIUMUserInfoGetEx](#) function. Note that the extended function **UserInfoGetEx** retrieves more information from the user account than **SAPIUMUserInfoGet** does.

```
Integer UserInfoGet (
    UMSESHandle          Handle,
    UserHandle           UserHandle,
    BSTR                 ref UserLoginName,
    BSTR                 ref UserCN,
    BSTR                 ref EmailAddress,
    SAPI_UM_ENUM_USER_TYPE ref Kind,
    Integer              ref RightMask,
    Integer              ref UpdateTime,
    BSTR                 ref Guid
)
```

## UserInfoGetEx

This function corresponds to the SAPIUM [SAPIUMUserInfoGetEx](#) function.

```
Integer UserInfoGetEx (
    UMSESHandle          Handle,
    UserHandle           UserHandle,
    BSTR                 ref UserLoginName,
    BSTR                 ref UserCN,
    BSTR                 ref EmailAddress,
    SAPI_UM_ENUM_USER_TYPE ref Kind,
    Integer              ref RightMask,
    Integer              ref UpdateTime,
    BSTR                 ref Guid,
    SAPI_UM_ENUM_USER_ENROLLMENT_STATUS ref EnrollmentStatus,
    SAPI_UM_ENUM_USER_ENROLLMENT_REASON ref EnrollmentReason,
    SAPI_UM_ENUM_USER_LOGIN_STATUS      ref LoginStatus,
    Integer                ref Counter1,
    Integer                ref Counter2,
    Integer                ref Counter3,
    SAPI_UM_ENUM_USER_CERT_STATUS_TYPE  ref CertStatus,
    SAPI_UM_ENUM_PENDING_REQUEST_STATUS_TYPE ref CertRequestStatus
)
```

## UserGetByLoginName

This function corresponds to the SAPIUM [SAPIUMUserGetByLoginName](#) function.

```
Integer UserGetByLoginName (
    UMSESHandle          Handle,
    BSTR                 UserLoginName,
    UserHandle           ref UserHandle
)
```

## UserTechIDGet

This function corresponds to the SAPIUM [SAPIUMUserTechIDGet](#) function.

```
Integer UserTechIDGet (
    UMSESHandle          Handle,
    UserHandle           UserHandle,
    long                 out UserTechID
)
```

## ResetPasswordCounter

This function corresponds to the SAPIUM [SAPIUMResetPasswordCounter](#) function.

```
Integer ResetPasswordCounter (
    UMSESHandle          Handle,
    BSTR                 UserLoginName
)
```

## GroupGetByUserGUID

This function corresponds to the SAPIUM [SAPIUMGroupGetByUserGUID](#) function.

```
Integer GroupGetByUserGUID (
    UMSESHandle          Handle,
    BSTR                 ref Guid,
    GroupRecord          out GroupRecord
)
```

## GroupAdd

This function corresponds to the SAPIUM [SAPIUMGroupAdd](#) function.

```
Integer GroupAdd (
    UMSESHandle          Handle,
    GroupRecord          GroupRecord,
    long                 out GroupTechID
)
```

## GroupUpdate

This function corresponds to the SAPIUM [SAPIUMGroupUpdate](#) function.

```
Integer GroupUpdate (
    UMSESHandle          Handle,
    GroupRecord          GroupRecord
)
```

## GroupSetStatusByTechID

This function corresponds to the SAPIUM [SAPIUMGroupSetStatusByTechID](#) function

```
Integer GroupSetStatusByTechID (
    UMSESHandle          Handle,
    long                 GroupTechID,
    SAPI_UM_ENUM_GROUP_STATUS_TYPE GroupStatus
)
```

## GroupDelete

This function corresponds to the SAPIUM [SAPIUMGroupDelete](#) function.

```
Integer GroupDelete (
    UMSESHandle          Handle,
    long                 GroupTechID,
    Integer              UserOp
)
```

## GroupGetByTechID

This function corresponds to the SAPIUM [SAPIUMGroupGetByTechID](#) function.

```
Integer GroupGetByTechID (
    UMSESHandle          Handle,
    long                 GroupTechID,
    GroupRecord          out GroupRecord
)
```

## GroupGetByName

This function corresponds to the SAPIUM [SAPIUMGroupGetByName](#) function.

```
Integer GroupGetByTechID (
    UMSESHandle          Handle,
    BSTR                 GroupName,
    GroupRecord          out GroupRecord
)
```

## GroupsEnumInit

This function corresponds to the SAPIUM [SAPIUMGroupsEnumInit](#) function.

```
Integer GroupsEnumInit (
    UMSESHandle          Handle,
    GroupContext         GroupsEnumContext,
    SAPI_UM_ENUM_GROUPS_ORDER_TYPE GroupsOrderType,
    BSTR                 StartValue,
    Integer              StartNumberValue,
    Integer              OrderDescending
)
```

## GroupsEnumCont

This function corresponds to the SAPIUM [SAPIUMGroupsEnumCont](#) function.

Please note that this function returns a single group record. In this it differs from the SAPIUM [SAPIUMGroupsEnumCont](#) function, which returns a bunch of group records for every call.

```
Integer GroupsEnumCont (
    UMSESHandle          Handle,
    GroupContext         GroupsEnumContext,
    GroupRecord          out GroupRecord
)
```

## GroupExtDataUpdate

This function corresponds to the SAPIUM [SAPIUMGroupUpdate](#) function.

```
Integer GroupExtDataUpdate (
    UMSESHandle          Handle,
    long                 GroupTechID,
    SAPI_UM_ENUM_GROUP_VALUE_NAME ValueType,
    Integer              ref IntVal,
    BSTR                 StrVal
)
```

## GroupExtDataGet

This function corresponds to the SAPIUM [SAPIUMGroupUpdate](#) function.

```
Integer GroupExtDataGet (
    UMSESHandle          Handle,
    long                 GroupTechID,
    SAPI_UM_ENUM_GROUP_VALUE_NAME ValueType,
    Integer              ref IntVal,
    BSTR                 ref StrVal
)
```

## SCPSet

This function corresponds to the SAPIUM [SAPIUMSCPSet](#) function.

```
Integer SCPSet (
    BSTR                 Address
)
```

## Appendices

---

The following sections provide additional supplementary information for the CoSign Signature Local signing, verifying, and user management API.



## Appendix A: CoSign Signature Local – Signing/Verifying Structures

This appendix describes the structures that can be passed as parameters to the CoSign Signature Local functions.

### SAPI\_CONTEXT

SAPI\_CONTEXT is a context value that is used by all the continuous operations. It must be passed to all the relevant functions without being changed by the application. Any change to SAPI\_CONTEXT can lead to inconsistent and unexpected results.

```
typedef struct _SAPI_CONTEXT {  
    SAPI_ENUM_CONTEXT_TYPE    ContextType;  
    void                      *OpaqueData;  
} SAPI_CONTEXT;
```

#### **Members**

##### *ContextType*

The type of the context. See [SAPI\\_ENUM\\_CONTEXT\\_TYPE](#) for possible values.

##### *OpaqueData*

Data that is being used internally by CoSign Signature Local. This data must not be changed by the application.

#### **See Also**

[SAPIContextRelease](#), [SAPICAInfoGetInit](#), [SAPICAInfoGetCont](#), [SAPISignatureFieldEnumInit](#),  
[SAPISignatureFieldEnumCont](#), [SAPICertificatesEnumInit](#), [SAPICertificatesEnumCont](#),  
[SAPIBufferSignInit](#), [SAPIBufferSignCont](#), [SAPIBufferSignEndEx](#), [SAPIBufferSignEx](#),  
[SAPISigningContextPKCS7BlobLenGet](#), [SAPIBufferVerifySignatureInit](#),  
[SAPIBufferVerifySignatureCont](#), [SAPIBufferVerifySignatureEnd](#)

## SAPI\_SF\_TIME\_FORMAT\_STRUCT

SAPI\_SF\_TIME\_FORMAT\_STRUCT is a structure that contains the information about the time format in a signature field.

```
typedef struct {  
    SAPI_ENUM_EXTENDED_TIME_FORMAT    ExtTimeFormat;  
    SAPI_WCHAR                        DateFormat[DATE_FORMAT_MAX_LEN];  
    SAPI_WCHAR                        TimeFormat[TIME_FORMAT_MAX_LEN];  
} SAPI_SF_TIME_FORMAT_STRUCT;
```

### Members

#### *ExtTimeFormat*

Extended time format. See [SAPI\\_ENUM\\_EXTENDED\\_TIME\\_FORMAT](#) for possible values.

#### *DateFormat*

The format string used to form the date string. The format string must be null terminated.

Use the following elements to construct a format string. If you use spaces to separate the elements in the format string, the spaces appear in the same location in the output string. The letters are case sensitive (for example, "MM" not "mm"). Characters in the format string that are enclosed in single quotation marks appear unchanged and in the same location in the output string.

Picture	Meaning
d	Day of the month as digits with no leading zero for single-digit days.
dd	Day of the month as digits with leading zero for single-digit days.
ddd	Day of the week as a three-letter abbreviation. The function uses the LOCALE_SABBREVDAYNAME value associated with the specified locale.
dddd	Day of the week as its full name. The function uses the LOCALE_SDAYNAME value associated with the specified locale.
M	Month as digits with no leading zero for single-digit months.
MM	Month as digits with leading zero for single-digit months.
MMM	Month as a three-letter abbreviation. The function uses the LOCALE_SABBREVMONTHNAME value associated with the specified locale.
MMMM	Month as its full name. The function uses the LOCALE_SMONTHNAME value associated with the specified locale.
y	Year as last two digits, but with no leading zero for single-digit years.
yy	Year as last two digits, but with a leading zero for single-digit years.
yyyy	Year represented by full four digits.
gg	Period/era string. The function uses the CAL_SERASTRING value associated with the specified locale. This element is ignored if the date to be formatted does not have an associated era or period string.

For example, to display the date string

"Wed, Aug 31 94"

use the following format string:

"ddd', ' MMM dd yy"

### *TimeFormat*

The format string to use to form the time string.

Use the following elements to construct a format string. If you use spaces to separate the elements in the format string, the spaces appear in the same location in the output string. The letters must be in uppercase or lowercase as shown (for example, "ss", not "SS"). Characters in the format string that are enclosed in single quotation marks appear unchanged and in the same location in the output string.

Picture	Meaning
h	Hours with no leading zero for single-digit hours; 12-hour clock.
hh	Hours with a leading zero for single-digit hours; 12-hour clock.
H	Hours with no leading zero for single-digit hours; 24-hour clock.
HH	Hours with a leading zero for single-digit hours; 24-hour clock.
m	Minutes with no leading zero for single-digit minutes.
mm	Minutes with a leading zero for single-digit minutes.
s	Seconds with no leading zero for single-digit seconds.
ss	Seconds with a leading zero for single-digit seconds.
t	One character time-marker string, such as A or P.
tt	Multicharacter time-marker string, such as AM or PM.

For example, to get the time string

"11:29:40 PM"

use the following format string:

"hh' :'mm' :'ss tt"

### **See Also**

[SAPI SIG FIELD SETTINGS](#)

## SAPI\_SIG\_FIELD\_SETTINGS

SAPI\_SIG\_FIELD\_SETTINGS is a structure that contains all the information that can be set for a signature field. Most of the information in this structure relates to the appearance of the field (both before and after it is signed).

```
typedef struct {
    unsigned long                StructLen;
    SAPI_WCHAR                   Name[SIGNATURE_FIELD_NAME_MAX_LEN];
    SAPI_ENUM_DEPENDENCY_MODE    DependencyMode;
    SAPI_ENUM_SIGNATURE_TYPE     SignatureType;
    long                          Page;
    long                          X;
    long                          Y;
    long                          Height;
    long                          Width;
    unsigned long                AppearanceMask;
    unsigned long                LabelsMask;
    SAPI_SF_TIME_FORMAT_STRUCT    TimeFormat;
    SAPI_WCHAR                   EmptyFieldLabel[EMPTY_FIELD_LABEL_MAX_LEN];
    SAPI_BOOL                    Invisible;
    unsigned long                Flags;
    void                          *ExtendedInfo;
} SAPI_SIG_FIELD_SETTINGS, *PSAPI_SIG_FIELD_SETTINGS;
```

### Members

#### *StructLen*

The length of this structure. This length must be set by the application prior to calling any function with this structure.

#### *Name*

The name of the signature field. This value is ignored when creating a new field, except when creating a new field in a PDF document with the AR\_PDF\_FLAG\_FIELD\_NAME\_SET flag.

#### *DependencyMode*

Indicates whether the signature field depends on other signature fields or not.

#### *SignatureType*

The signature type, see [SAPI\\_ENUM\\_SIGNATURE\\_TYPE](#) for possible values.

#### *Page*

The page in the document where the signature field is located.

In Word, you can use -1 to indicate the last page.

In PDF, you can use -1 to indicate the last page, -2 to indicate the penultimate page, etc.

#### *X*

The X coordinate of the signature field rectangle.

#### *Y*

The Y coordinate of the signature field rectangle.

#### *Height*

The height of the signature field rectangle.

### *Width*

The width of the signature field rectangle.

### *AppearanceMask*

A bit mask indicating which information is displayed when the signature field is signed (graphical image, signer, reason, date). See the mask values description in [SAPI\\_ENUM\\_DRAWING\\_ELEMENT](#).

### *LabelsMask*

A bit mask indicating which labels (for signed by, reason and date) are displayed when the signature field is signed. This bit mask uses the same constants as AppearanceMask.

### *TimeFormat*

The date and time format of the signing time displayed for the signed field. See [SAPI\\_SF\\_TIME\\_FORMAT\\_STRUCT](#) for more information.

### *EmptyFieldLabel*

This field is only relevant for Word documents. It represents the string displayed on an unsigned signature field rectangle.

### *Invisible*

A flag indicating whether the signature is invisible (true) or visible (false).

### *Flags*

Bit mask flags. For more information, refer to the following sections in Appendix D: [Flags when Using a Signature Field in Word Documents](#), [Flags when Using a Signature Field in PDF Documents](#) and [Flags when Creating a Signature Field in TIFF Files](#).

Note that some signature field flags are passed as part of the [SAPISignatureFieldCreateEx](#) or

SAPISignatureFieldCreateSignEx functions, while others are passed as part of this SAPI\_SIG\_FIELD\_SETTINGS structure.

The relevant sections in [Appendix D](#) note which flags are passed as part of a function and which are passed as part of the signature field structure.

*ExtendedInfo*

Pointer to an extended information value.

Use this extended information to maintain a list of custom fields.

For information on custom fields, refer to [SAPI\\_CUSTOM\\_FIELD\\_ELEMENTS\\_STRUCT](#).

**See Also**

[SAPISignatureFieldInfoGet](#), [SAPISignatureFieldCreateEx](#)

## SAPI\_GRAPHIC\_IMAGE\_STRUCT

SAPI\_GRAPHIC\_IMAGE\_STRUCT is a structure that contains the graphical image data and format.

This structure is used both for getting the graphical image from a signed signature field and for getting the graphical image information that is stored in DocuSign Signature Appliance by calling [SAPIGraphicSigImageGet](#). [SAPIGraphicSigImageGet](#) is an obsolete function. Use the [SAPI\\_GR\\_IMG\\_INFO](#) structure and [SAPIGraphicSigImageInfoGet](#) function to get graphical image information from DocuSign Signature Appliance. Only [SAPISignatureFieldInfoGet](#) uses [SAPI\\_GRAPHIC\\_IMAGE\\_STRUCT](#).

Note that **GraphicImage** is allocated by the [SAPIGraphicSigImageGet](#) function and must be released by the [SAPIStructRelease](#) function.

Note that if [SAPI\\_GRAPHIC\\_IMAGE\\_STRUCT](#) is returned as part of the signed signature field information, it is automatically released when the [SAPI\\_SIGNED\\_FIELD\\_INFO](#) structure is released.

```
typedef struct {
    unsigned char          *GraphicImage;
    long                  GraphicImageLen;
    SAPI_ENUM_GRAPHIC_IMAGE_FORMAT GraphicImageFormat;
    long                  Width;
    long                  Height;
} SAPI_GRAPHIC_IMAGE_STRUCT, *PSAPI_GRAPHIC_IMAGE_STRUCT;
```

### Members

#### GraphicImage

Pointer to the graphical image data. The format of the data must comply with the value of **GraphicImageFormat**.

#### GraphicImageLen

The length in bytes of **GraphicImage**.

#### GraphicImageFormat

The format of the graphical image in **GraphicImage**. See [SAPI\\_ENUM\\_GRAPHIC\\_IMAGE\\_FORMAT](#) for possible values.

#### Width

The width in pixels of the graphical image.

#### Height

The height in pixels of the graphical image.

### See Also

[SAPIGraphicSigImageGet](#), [SAPIGraphicSigImageSet](#), [SAPISignatureFieldInfoGet](#), [SAPIStructRelease](#), [SAPIGraphicSigImageInfoGet](#)

## SAPI\_GR\_IMG\_INFO

SAPI\_GR\_IMG\_INFO is a structure that contains data related to a graphical signature image. This structure is used when creating, updating, or retrieving information about a new graphical image.

```
typedef struct {
    unsigned long          StructLen;
    SAPI_WCHAR            szGraphicImageName[GR_SIG_NAME_MAX_LEN];
    unsigned long         DataFormat;
    unsigned char         *GraphicImage;
    unsigned long         GraphicImageLen;
    SAPI_ENUM_GRAPHIC_IMAGE_FORMAT ImageConvertedFormat;
    long                  Width;
    long                  Height;
    long                  DataType;
} SAPI_GR_IMG_INFO, *PSAPI_GR_IMG_INFO;
```

### Members

#### *StructLen*

The size of the structure. This field must be set prior to calling to **any** function that gets this structure as a parameter.

#### *szGraphicImageName*

The name of the graphical image object in wide chars.

#### *DataFormat*

Bit flag value that indicates the format of the graphical image object. You may set one or more bits. For the possible settings for *DataFormat*, see [Graphical Signature Image Related Flags](#).

#### *GraphicImage*

The actual data of the graphical image. This pointer can be either allocated by the calling application or by CoSign Signature Local, see [SAPIGraphicSigImageInfoGet](#) for more information.

#### *GraphicImageLen*

The size in bytes that is allocated for *GraphicImage*, or the actual size of the data that was returned in *GraphicImage*.

#### *ImageConvertedFormat*

Indicates the format of the graphical image as was requested when this structure was filled when [SAPIGraphicSigImageInfoGet](#) was called.

#### *Width*

For known image formats (BMP, JPG), this member contains the width of the image in pixels.

#### *Height*

For known image formats (BMP, JPG), this member contains the height of the image in pixels.

#### *DataType*

The type of graphical image. The supported types are Graphical Signature, Initials, and Logo. For the available types, refer to [Graphical Signature Image Related Flags](#) below.

## SAPI\_FILETIME

SAPI\_FILETIME is a structure containing date/time information.

```
typedef struct {  
    struct {  
        unsigned long LowDateTime;  
        unsigned long HighDateTime;  
    } LocalTime;  
    long    GMTOffset;  
} SAPI_FILETIME;
```

### ***Members***

#### *LowDateTime*

The low-order part of the date-time.

#### *HighDateTime*

The high-order part of the date-time.

#### *GMTOffset*

The GMT offset in minutes.

## SAPI\_SIGNED\_FIELD\_INFO

SAPI\_SIGNED\_FIELD\_INFO is a structure that contains all the information related to a signed signature field.

```
typedef struct {
    unsigned long          StructLen;
    SAPI_LPWSTR           SignerName;
    SAPI_BOOL             IsSigned;
    unsigned char         *Certificate;
    long                  CertificateLen;
    SAPI_FILETIME         SignatureTime;
    SAPI_LPWSTR           Reason;
    PSAPI_GRAPHIC_IMAGE_STRUCT GraphicImageStruct;
    SAPI_LPWSTR           DependencyString;
    PSAPI_GRAPHIC_IMAGE_STRUCT GraphicLogoStruct;
} SAPI_SIGNED_FIELD_INFO, *PSAPI_SIGNED_FIELD_INFO;
```

### **Members**

#### *StructLen*

The length of this structure. This length must be set by the application prior to calling any function with this structure.

#### *SignerName*

The signer's name.

#### *IsSigned*

Indicates whether the signature field is signed. This is the only field returned for unsigned signature fields.

#### *Certificate*

The ASN1 value of the certificate used for signing.

#### *CertificateLen*

The certificate data length.

#### *SignatureTime*

When the signing operation was performed.

#### *Reason*

The reason for signing (only if entered during the signature operation).

#### *GraphicImageStruct*

The graphical image attached to the signed field.

#### *DependencyString*

A wide char string containing all the names of the signature fields that were signed prior to the current field, and which the current field depends on. The names are separated by a null character and the string is terminated with double null characters.

#### *GraphicLogoStruct*

The Logo image attached to the signed field.

***See Also***

[SAPISignatureFieldInfoGet](#), [SAPIStructRelease](#)

## SAPI\_INFO\_STRUCT

**SAPI\_INFO\_STRUCT** is a structure that contains SAPI lib general information, such as version and installation mode (verify and sign mode or verify-only mode).

```
typedef struct {  
    SAPI_BOOL                VerifyModeOnly;  
    unsigned long            MajorVersion;  
    unsigned long            MinorVersion;  
} SAPI_INFO_STRUCT, *PSAPI_INFO_STRUCT;
```

### **Members**

#### *VerifyModeOnly*

Indicates whether the current CoSign Signature Local installation can be used for signature verification only (true), or for both signing and verifying (false).

#### *MajorVersion*

CoSign Signature Local major version number.

#### *MinorVersion*

CoSign Signature Local minor version number.

### **See Also**

[SAPILibInfoGet](#)

## SAPI\_CERT\_STATUS\_STRUCT

SAPI\_CERT\_STATUS\_STRUCT is a structure that contains the certificate status as returned from a signature verification function. The structure includes two fields: the Operating System status as is, and the CoSign Signature Local interpretation of the OS status.

```
typedef struct {  
    SAPI_ENUM_CERT_STATUS          SAPICertStatus;  
    unsigned long                   OSCertStatus;  
} SAPI_CERT_STATUS_STRUCT, *PSAPI_CERT_STATUS_STRUCT;
```

### **Members**

#### *SAPICertStatus*

The CoSign Signature Local interpretation of the certificate status as returned by the OS. See [SAPI\\_ENUM\\_CERT\\_STATUS](#) for possible values.

#### *OSCertStatus*

The certificate status as returned by the operating system.

### **See Also**

[SAPISignatureFieldVerify](#), [SAPIBufferVerifySignature](#), [SAPIBufferVerifySignatureEnd](#)

## SAPI\_CUSTOM\_FIELD\_ELEMENTS\_STRUCT

SAPI\_CUSTOM\_FIELD\_ELEMENTS\_STRUCT is a structure that contains the list of elements that are included in the signature field. The struct of elements points to an internal structure of a single element called SAPI\_CUSTOM\_FIELD\_ELEMENT\_STRUCT.

```
typedef struct {
    int                ID;
    SAPI_ENUM_DATA_TYPE Type;
    unsigned char     *Value;
    unsigned long      ValueLen;
} SAPI_CUSTOM_FIELD_ELEMENT_STRUCT;

typedef struct {
    int                StructLen;
    int                Version;
    int                NumOfElements;
    SAPI_CUSTOM_FIELD_ELEMENT_STRUCT CustomFieldsArray[MAX_NUM_OF_CUSTOM_FIELDS];
} SAPI_CUSTOM_FIELD_ELEMENTS_STRUCT, *PSAPI_CUSTOM_FIELD_ELEMENTS_STRUCT;
```

### Members of SAPI\_CUSTOM\_FIELD\_ELEMENT\_STRUCT

#### *ID*

The ID of the custom field.

#### *Type*

The type of the custom field. See SAPI\_ENUM\_DATA\_TYPE for possible values.

#### *Value*

The value of the custom field.

#### *ValueLen*

The length of the custom field.

### Members of SAPI\_CUSTOM\_FIELD\_ELEMENTS\_STRUCT

#### *StructLen*

The length of this structure. This length must be set by the application prior to calling any function with this structure.

#### *Version*

Internal use.

#### *NumOfElements*

The number of actual elements.

#### *CustomFieldsArray*

An array of the custom fields.

### **See Also**

[SAPISignatureFieldVerify](#), [SAPIBufferVerifySignature](#), [SAPIBufferVerifySignatureEnd](#)

## SAPI\_TOKEN\_VERSION

SAPI\_TOKEN\_VERSION contains version information of a software, firmware or hardware version.

```
typedef struct {  
    unsigned long          Major;  
    unsigned long          Minor;  
} SAPI_TOKEN_VERSION;
```

### *Members*

#### **Major**

The major version number.

#### **Minor**

The minor version number.

## SAPI\_TOKEN\_INFO\_STRUCT

SAPI\_TOKEN\_INFO\_STRUCT is returned by the [SAPIGetTokenInfo](#) function and contains all relevant information about the appliance used by the SAPI Session.

```
typedef struct {
    SAPI_WCHAR                TokenID[TOKEN_ID_MAX_LEN];
    SAPI_TOKEN_VERSION        Firmware;
    SAPI_TOKEN_VERSION        Hardware;
    SAPI_WCHAR                SerialNumber[SERVER_SERIAL_NUMBER_LEN];
    unsigned long             TokenTime;
    int                        InstallStatus;
    SAPI_ENUM_SERVER_KIND     ServerKind;
    SAPI_ENUM_DIRECTORY_KIND DirectoryKind;
    unsigned long             SubDirectoryKind;
    SAPI_ENUM_AUTH_MODE       AuthMode;
    SAPI_ENUM_AUTH_MODE       AuthMode2;
    long                       ClusterId;
    void                       *ExtendedInfo;
} SAPI_CERT_STATUS_STRUCT, *PSAPI_CERT_STATUS_STRUCT;
```

### **Members**

#### **TokenID**

The identification of the appliance.

#### **Firmware**

The firmware version.

#### **Hardware**

The hardware version.

#### **SerialNumber**

The serial number of the appliance.

#### **TokenTime**

The current time of the appliance. For more information, refer to [SAPITimeGet](#).

#### **InstallStatus**

The installation status.

#### **ServerKind**

The Server type, according to the [SAPI\\_ENUM\\_SERVER\\_KIND](#) enumerated type.

#### **DirectoryKind**

The type of directory used, according to the [SAPI\\_ENUM\\_DIRECTORY\\_KIND](#) enumerated type.

#### **SubDirectoryKind**

The subdirectory type used (relevant for an LDAP directory).

#### **AuthMode**

The authentication mode used.

**AuthMode2**

The authentication mode used for Prompt For Sign.

**ClusterID**

The identification of the Cluster. For more information contact ARX Support.

**ExtendedInfo**

Extended Info. For more information contact ARX Support.

***See Also***

[SAPIGetTokenInfo](#)

## Appendix B: CoSign Signature Local – Signing/Verifying TYPEDEFS

This appendix describes the CoSign Signature Local type definitions that are not structures.

### SAPI\_SES\_HANDLE

The type definition of the SAPI session handle.

```
typedef void *SAPI_SES_HANDLE;
```

### SAPI\_GR\_IMG\_HANDLE

The type definition of a graphical image handle. In this version, a graphical image handle is not being used.

```
typedef void *SAPI_GR_IMG_HANDLE;
```

### SAPI\_SIG\_FIELD\_HANDLE

The type definition of a signature field handle.

```
typedef void *SAPI_SIG_FIELD_HANDLE;
```

### SAPI\_CERT\_HANDLE

The type definition of a certificate handle.

```
typedef void *SAPI_CERT_HANDLE;
```

### SAPI\_FILE\_HANDLE

The type definition of a file handle.

```
typedef void *SAPI_FILE_HANDLE;
```

### SAPI\_BOOL

The CoSign Signature Local definition for a Boolean variable.

```
typedef int SAPI_BOOL;
```

### SAPI\_PROGRESS\_CALLBACK

The definition of the progress bar callback function, called by CoSign Signature Local when signing and verifying files.

```
typedef void (SAPI_WINAPI *SAPI_PROGRESS_CALLBACK) (  
    SAPI_ENUM_PROGRESS_CALLBACK_OPERATION    Operation,  
    void *Arg  
);
```

When the [SAPISignatureFieldSign](#) or [SAPISignatureFieldVerify](#) functions are called, the signing, verifying and hashing operations processed by CoSign Signature Local, send the BEGIN event when starting the operation. They then divide the file into several blocks of data, and send the blocks one by one for signing/hashing or verifying, depending on the function called. Each time a block is sent for processing, the CONT event is sent to the callback function. When data processing is complete, the END event is called.

Following is a table describing the arguments passed for each operation.

Operation	Arg value
SAPI_ENUM_PROGRESS_CALLBACK_SIGN_BEGIN	The size of the file being signed.
SAPI_ENUM_PROGRESS_CALLBACK_SIGN_CONT	The number of additional bytes sent for signing.
SAPI_ENUM_PROGRESS_CALLBACK_SIGN_END	The signature data size.
SAPI_ENUM_PROGRESS_CALLBACK_VERIFY_BEGIN	The size of the file being verified.
SAPI_ENUM_PROGRESS_CALLBACK_VERIFY_CONT	The number of additional bytes sent for verification.
SAPI_ENUM_PROGRESS_CALLBACK_VERIFY_END	
SAPI_ENUM_PROGRESS_CALLBACK_HASH_BEGIN	The number of bytes to be hashed.
SAPI_ENUM_PROGRESS_CALLBACK_HASH_CONT	The number of additional bytes sent for hashing.
SAPI_ENUM_PROGRESS_CALLBACK_HASH_END	The length of the hash result.

## Appendix C: CoSign Signature Local – Signing/Verifying ENUMS

This appendix describes the type definitions for the enumerated values (`typedef enum {...}`).

### SAPI\_ENUM\_DRAWING\_ELEMENT

The elements to be displayed in a signed signature field. To draw more than one element, logically OR all the values of the desired elements.

```
typedef enum SAPI_ENUM_DRAWING_ELEMENT{
    SAPI_ENUM_DRAWING_ELEMENT_GRAPHICAL_IMAGE    = 0x00000001,
    SAPI_ENUM_DRAWING_ELEMENT_SIGNED_BY          = 0x00000002,
    SAPI_ENUM_DRAWING_ELEMENT_REASON              = 0x00000004,
    SAPI_ENUM_DRAWING_ELEMENT_TIME                = 0x00000008,
    SAPI_ENUM_DRAWING_ELEMENT_TITLE               = 0x00000020,
    SAPI_ENUM_DRAWING_ELEMENT_ADDITIONAL_TXT     = 0x00000020,
    SAPI_ENUM_DRAWING_ELEMENT_LOGO                = 0x00000040,
    SAPI_ENUM_DRAWING_ELEMENT_SUGGESTED_TITLE    = 0x00000080,
    SAPI_ENUM_DRAWING_ELEMENT_INITIALS           = 0x40000000
} SAPI_ENUM_DRAWING_ELEMENT;
```

#### **ENUM values**

##### **SAPI\_ENUM\_DRAWING\_ELEMENT\_GRAPHICAL\_IMAGE**

For displaying the graphical image (if exists), in the signed field.

##### **SAPI\_ENUM\_DRAWING\_ELEMENT\_SIGNED\_BY**

Display the name of the signer, in the signed field.

##### **SAPI\_ENUM\_DRAWING\_ELEMENT\_REASON**

Display the reason for signing, in the signed field.

##### **SAPI\_ENUM\_DRAWING\_ELEMENT\_TIME**

Display when the file was signed, in the signed field.

##### **SAPI\_ENUM\_DRAWING\_ELEMENT\_TITLE**

Display the user's title in the signed field.

##### **SAPI\_ENUM\_DRAWING\_ELEMENT\_ADDITIONAL\_TXT**

Display additional text in the signed field. The value of this element is identical to the value of SAPI\_ENUM\_DRAWING\_ELEMENT\_TITLE.

##### **SAPI\_ENUM\_DRAWING\_ELEMENT\_LOGO**

Displays a Logo(if one exists), in the signed field.

##### **SAPI\_ENUM\_DRAWING\_ELEMENT\_SUGGESTED\_TITLE**

Not Implemented.

##### **SAPI\_ENUM\_DRAWING\_ELEMENT\_INITIALS**

This enumerated value must be set only if SAPI\_ENUM\_DRAWING\_ELEMENT\_GRAPHICAL\_IMAGE is not set.

## SAPI\_ENUM\_PROGRESS\_CALLBACK\_OPERATION

The operation events sent to the progress bar callback function, while the file signing and verifying functions are being processed. See [SAPI\\_PROGRESS\\_CALLBACK](#) for more details.

```
typedef enum SAPI_ENUM_PROGRESS_CALLBACK_OPERATION {
    SAPI_ENUM_PROGRESS_CALLBACK_OPERATION_NONE    = 0,
    SAPI_ENUM_PROGRESS_CALLBACK_SIGN_BEGIN        = 1,
    SAPI_ENUM_PROGRESS_CALLBACK_SIGN_CONT         = 2,
    SAPI_ENUM_PROGRESS_CALLBACK_SIGN_END          = 3,
    SAPI_ENUM_PROGRESS_CALLBACK_VERIFY_BEGIN      = 4,
    SAPI_ENUM_PROGRESS_CALLBACK_VERIFY_CONT       = 5,
    SAPI_ENUM_PROGRESS_CALLBACK_VERIFY_END        = 6,
    SAPI_ENUM_PROGRESS_CALLBACK_HASH_BEGIN        = 7,
    SAPI_ENUM_PROGRESS_CALLBACK_HASH_CONT         = 8,
    SAPI_ENUM_PROGRESS_CALLBACK_HASH_END         = 9
} SAPI_ENUM_PROGRESS_CALLBACK_OPERATION;
```

### **ENUM values**

**SAPI\_ENUM\_PROGRESS\_CALLBACK\_OPERATION\_NONE**

Not in use.

**SAPI\_ENUM\_PROGRESS\_CALLBACK\_SIGN\_BEGIN**

The signing operation has begun.

**SAPI\_ENUM\_PROGRESS\_CALLBACK\_SIGN\_CONT**

An additional buffer of data read from the file was sent for signing.

**SAPI\_ENUM\_PROGRESS\_CALLBACK\_SIGN\_END**

The file signing operation is completed.

**SAPI\_ENUM\_PROGRESS\_CALLBACK\_VERIFY\_BEGIN**

The verifying operation has begun.

**SAPI\_ENUM\_PROGRESS\_CALLBACK\_VERIFY\_CONT**

An additional buffer of data read from the file was sent for verifying.

**SAPI\_ENUM\_PROGRESS\_CALLBACK\_VERIFY\_END**

The file verifying operation is completed.

**SAPI\_ENUM\_PROGRESS\_CALLBACK\_HASH\_BEGIN**

Not in use.

**SAPI\_ENUM\_PROGRESS\_CALLBACK\_HASH\_CONT**

Not in use.

**SAPI\_ENUM\_PROGRESS\_CALLBACK\_HASH\_END**

Not in use.

## SAPI\_ENUM\_CONTEXT\_TYPE

Context type values. All the continuous operations share the same definition for their context (SAPI\_CONTEXT), but are distinguished by the value of ContextType. ContextType can be any of the following values:

```
typedef enum SAPI_ENUM_CONTEXT_TYPE {
    SAPI_ENUM_CONTEXT_TYPE_NONE           = 0,
    SAPI_ENUM_CONTEXT_TYPE_CA             = 1,
    SAPI_ENUM_CONTEXT_TYPE_ENUM_CERT     = 2,
    SAPI_ENUM_CONTEXT_BUFF_SIGN           = 3,
    SAPI_ENUM_CONTEXT_BUFF_VERIFY         = 4,
    SAPI_ENUM_CONTEXT_BUFF_HASH           = 5,
    SAPI_ENUM_CONTEXT_TYPE_ENUM_SF        = 6,
    SAPI_ENUM_CONTEXT_TYPE_ENUM_GR_IMG    = 7,
    SAPI_ENUM_CONTEXT_TICKET_CHECK        = 8,
    SAPI_ENUM_CONTEXT_SF_LOCATOR          = 9,
    SAPI_ENUM_CONTEXT_TYPE_SF_COMBINED    = 10
} SAPI_ENUM_CONTEXT_TYPE;
```

### **ENUM values**

**SAPI\_ENUM\_CONTEXT\_TYPE\_NONE**

Not in use.

**SAPI\_ENUM\_CONTEXT\_TYPE\_CA**

The context for retrieving the CA information.

**SAPI\_ENUM\_CONTEXT\_TYPE\_ENUM\_CERT**

The context for enumerating certificates.

**SAPI\_ENUM\_CONTEXT\_BUFF\_SIGN**

The context for signing buffers.

**SAPI\_ENUM\_CONTEXT\_BUFF\_VERIFY**

The context for verifying a signature on buffers.

**SAPI\_ENUM\_CONTEXT\_BUFF\_HASH**

The context for hashing data.

**SAPI\_ENUM\_CONTEXT\_TYPE\_ENUM\_SF**

The context for enumerating signature fields.

**SAPI\_ENUM\_CONTEXT\_TYPE\_ENUM\_GR\_IMG**

The context for enumerating graphical images.

**SAPI\_ENUM\_CONTEXT\_TICKET\_CHECK**

Contact ARX for more information.

**SAPI\_ENUM\_CONTEXT\_SF\_LOCATOR**

The context of enumerating PDF field locator strings for positioning a signature field.

**SAPI\_ENUM\_CONTEXT\_TYPE\_SF\_COMBINED**

The context for enumerating signature fields, and signature fields based on locators.

## SAPI\_ENUM\_EXTENDED\_TIME\_FORMAT

The time format in the signature field is determined by the date and time format strings (e.g., “dd/mm/yyyy” and “hh:mm:ss”), but there are some additional extended attributes that are determined by the following values:

```
typedef enum SAPI_ENUM_EXTENDED_TIME_FORMAT {  
    SAPI_ENUM_EXTENDED_TIME_FORMAT_NONE    = 0,  
    SAPI_ENUM_EXTENDED_TIME_FORMAT_GMT     = 1,  
    SAPI_ENUM_EXTENDED_TIME_FORMAT_SYSTEM  = 2  
} SAPI_ENUM_EXTENDED_TIME_FORMAT;
```

### *ENUM values*

#### *SAPI\_ENUM\_EXTENDED\_TIME\_FORMAT\_NONE*

Not in use.

#### *SAPI\_ENUM\_EXTENDED\_TIME\_FORMAT\_GMT*

The difference of the local time zone from GMT time is added to the date and time string.

#### *SAPI\_ENUM\_EXTENDED\_TIME\_FORMAT\_SYSTEM*

The settings for the date and time format are taken from the system settings, and the date and time format strings are ignored. This value is not supported in the current version.

## SAPI\_ENUM\_DEPENDENCY\_MODE

The dependency mode of a single signature field.

```
typedef enum SAPI_ENUM_DEPENDENCY_MODE {  
    SAPI_ENUM_DEPENDENCY_MODE_NONE      = 0,  
    SAPI_ENUM_DEPENDENCY_MODE_DEPENDENT = 1,  
    SAPI_ENUM_DEPENDENCY_MODE_INDEPENDENT = 2,  
} SAPI_ENUM_DEPENDENCY_MODE;
```

### ***ENUM values***

#### ***SAPI\_ENUM\_DEPENDENCY\_MODE\_NONE***

Not in use.

#### ***SAPI\_ENUM\_DEPENDENCY\_MODE\_DEPENDENT***

The signature in the current signature field depends on all the previously-signed fields. This means that if a signature in one of the previously signed fields is cleared or recreated, the signature of the current field becomes invalid.

#### ***SAPI\_ENUM\_DEPENDENCY\_MODE\_INDEPENDENT***

The signature in the current signature field is independent. This means that clearing or resigning of other signature fields in the same document do not invalidate the current signature.

## SAPI\_ENUM\_SIGNATURE\_TYPE

The signature type of a specific signature field. The signature type influences both the appearance of the signature field and the signing algorithms being used.

```
typedef enum SAPI_ENUM_SIGNATURE_TYPE {  
    SAPI_ENUM_SIGNATURE_NONE                = 0,  
    SAPI_ENUM_SIGNATURE_DIGITAL              = 1,  
    SAPI_ENUM_SIGNATURE_E_HASH               = 2  
} SAPI_ENUM_SIGNATURE_TYPE;
```

### *ENUM values*

#### *SAPI\_ENUM\_SIGNATURE\_NONE*

Not in use.

#### *SAPI\_ENUM\_SIGNATURE\_DIGITAL*

Standard visible signature (PKCS#7, RSA and SHA1).

#### *SAPI\_ENUM\_SIGNATURE\_E\_HASH*

Electronic visible signature. Only hash algorithm is used. This type is not supported in the current version.

## SAPI\_ENUM\_GRAPHIC\_IMAGE\_FORMAT

The different formats in which the graphical image of the user's signature can be loaded and retrieved.

```
typedef enum SAPI_ENUM_GRAPHIC_IMAGE_FORMAT {  
    SAPI_ENUM_GRAPHIC_IMAGE_NONE           = 0,  
    SAPI_ENUM_GRAPHIC_IMAGE_PDF_LINE      = 1,  
    SAPI_ENUM_GRAPHIC_IMAGE_PDF_BMP      = 2,  
    SAPI_ENUM_GRAPHIC_IMAGE_BMP          = 3,  
    SAPI_ENUM_GRAPHIC_IMAGE_BMP_B64     = 4,  
    SAPI_ENUM_GRAPHIC_IMAGE_USER_DEFINED  = 5,  
    SAPI_ENUM_GRAPHIC_IMAGE_PDF_BMP2    = 6  
} SAPI_ENUM_GRAPHIC_IMAGE_FORMAT;
```

### *ENUM values*

*SAPI\_ENUM\_GRAPHIC\_IMAGE\_NONE*

Not in use.

*SAPI\_ENUM\_GRAPHIC\_IMAGE\_PDF\_LINE*

PDF format vectorial representation.

*SAPI\_ENUM\_GRAPHIC\_IMAGE\_PDF\_BMP*

PDF format BMP representation (monochrome).

*SAPI\_ENUM\_GRAPHIC\_IMAGE\_BMP*

BMP format.

*SAPI\_ENUM\_GRAPHIC\_IMAGE\_BMP\_B64*

BMP format, encoded as base 64.

*SAPI\_ENUM\_GRAPHIC\_IMAGE\_USER\_DEFINED*

The graphical image is stored and retrieved as is.

*SAPI\_ENUM\_GRAPHIC\_IMAGE\_PDF\_BMP2*

PDF format BMP representation. Unlike *SAPI\_ENUM\_GRAPHIC\_IMAGE\_PDF\_BMP*, this format can hold monochrome and colored BMP as well as JPG formats. This format also makes white pixels transparent.

## SAPI\_ENUM\_CA\_INFO\_TYPE

The information that can be retrieved from the CA.

```
typedef enum SAPI_ENUM_CA_INFO_TYPE {  
    SAPI_ENUM_CA_INFO_NONE           = 0,  
    SAPI_ENUM_CA_INFO_AIA            = 1,  
    SAPI_ENUM_CA_INFO_CRL            = 2  
} SAPI_ENUM_CA_INFO_TYPE;
```

### ***ENUM values***

*SAPI\_ENUM\_CA\_INFO\_NONE*

Not in use.

*SAPI\_ENUM\_CA\_INFO\_AIA*

The CA certificate.

*SAPI\_ENUM\_CA\_INFO\_CRL*

The Certificate Revocation List.

## SAPI\_ENUM\_FILE\_TYPE

The file types that can be processed by the CoSign Signature Local file functions.

```
typedef enum SAPI_ENUM_FILE_TYPE {  
    SAPI_ENUM_FILE_NONE           = 0,  
    SAPI_ENUM_FILE_WORD           = 1,  
    SAPI_ENUM_FILE_ADOBE          = 2,  
    SAPI_ENUM_FILE_TIFF           = 3,  
    SAPI_ENUM_FILE_DETACHED       = 4,  
    SAPI_ENUM_FILE_P7M            = 5,  
    SAPI_ENUM_FILE_XML            = 6,  
    SAPI_ENUM_FILE_OFFICE_XML_PACKAGE = 7,  
    SAPI_ENUM_FILE_INFOPATH_XML_FORM = 8  
} SAPI_ENUM_FILE_TYPE;
```

### ***ENUM values***

*SAPI\_ENUM\_FILE\_NONE*

Not in use.

*SAPI\_ENUM\_FILE\_WORD*

Word file.

*SAPI\_ENUM\_FILE\_ADOBE*

PDF file (Adobe).

*SAPI\_ENUM\_FILE\_TIFF*

TIFF file.

*SAPI\_ENUM\_FILE\_DETACHED*

Not supported in the current version.

*SAPI\_ENUM\_FILE\_P7M*

Not supported in the current version.

*SAPI\_ENUM\_FILE\_XML*

XML File. Supported from version 5.

*SAPI\_ENUM\_FILE\_OFFICE\_XML\_PACKAGE*

Office 2007 file type (.docx file or .xlsx file).

*SAPI\_ENUM\_FILE\_INFOPATH\_XML\_FORM*

InfoPath 2007/2010/2013 form (.xml file).

## SAPI\_ENUM\_CONF\_ID

The configuration values that can be set and retrieved.

```
typedef enum SAPI_ENUM_CONF_ID {
    SAPI_ENUM_FILE_NONE = 0,
    SAPI_ENUM_CONF_ID_REASON = 1,
    SAPI_ENUM_CONF_ID_SEL_CERT_TITLE = 2,
    SAPI_ENUM_CONF_ID_SEL_CERT_CMD = 3,
    SAPI_ENUM_CONF_ID_CHK_CRL_ENUM = 4,
    SAPI_ENUM_CONF_ID_CHK_CRL_VERIFY = 5,
    SAPI_ENUM_CONF_ID_VERIFY_CERT_SIGN = 6,
    SAPI_ENUM_CONF_ID_PROGRESS_CALLBACK = 7,
    SAPI_ENUM_CONF_ID_WORD_SF_FUNC = 8,
    SAPI_ENUM_CONF_ID_REASON_LABEL = 9,
    SAPI_ENUM_CONF_ID_DATE_LABEL = 10,
    SAPI_ENUM_CONF_ID_SIGNER_LABEL = 11,
    SAPI_ENUM_CONF_ID_CERT_SERIAL_ID = 12,
    SAPI_ENUM_CONF_ID_CERT_CHAIN_FLAGS = 13,
    SAPI_ENUM_CONF_ID_PREFERRED_TOKEN = 14,
    SAPI_ENUM_CONF_ID_CERT_SKI = 15,
    SAPI_ENUM_CONF_ID_CERT_SET_DEFAULT = 16,
    SAPI_ENUM_CONF_ID_GR_SIG_PREF_NAME = 17,
    SAPI_ENUM_CONF_ID_PDF_OWNER_PWD = 18,
    SAPI_ENUM_CONF_ID_PDF_USER_PWD = 19,
    SAPI_ENUM_CONF_ID_PDF_SF_FUNC = 20,
    SAPI_ENUM_CONF_ID_GMT_OFFSET = 21,
    SAPI_ENUM_CONF_ID_RET_EXT_FIELD_INFO = 22,
    SAPI_ENUM_CONF_ID_SECURED_TS_ENABLE = 23,
    SAPI_ENUM_CONF_ID_SECURED_TS_URL = 24,
    SAPI_ENUM_CONF_ID_SECURED_TS_USER = 25,
    SAPI_ENUM_CONF_ID_SECURED_TS_PWD = 26,
    SAPI_ENUM_CONF_ID_SECURED_TS_ADDITIONAL_BYTES = 27,
    SAPI_ENUM_CONF_ID_TS_CERT_STATUS = 28,
    SAPI_ENUM_CONF_ID_TITLE = 29,
    SAPI_ENUM_CONF_ID_ADDITIONAL_TEXT = 29,
    SAPI_ENUM_CONF_ID_PDF_ROAMING_ID_SRV = 30,
    SAPI_ENUM_CONF_ID_KEY_CONTAINER_NAME = 31,
    SAPI_ENUM_CONF_ID_SLOT_NUMBER = 32,
    SAPI_ENUM_CONF_ID_LOGO_PREF_NAME = 33,
    SAPI_ENUM_CONF_ID_LOCK_USER = 34,
    SAPI_ENUM_CONF_ID_EMBED_OCSP = 35,
    SAPI_ENUM_CONF_ID_OCSP_URL = 36,
    SAPI_ENUM_CONF_ID_CERT_PUBLIC = 37,
    SAPI_ENUM_CONF_ID_TIFF_BANNER_CREATE = 38,
    SAPI_ENUM_CONF_ID_TIFF_IGNORE_EMBEDDED_CLEAR = 39,
    SAPI_ENUM_CONF_ID_XADES_TEMPLATE = 40,
    SAPI_ENUM_CONF_ID_EXTENDED_VALIDATION = 41,
    SAPI_ENUM_CONF_ID_PDF_FONT_PATH = 42,
    SAPI_ENUM_CONF_ID_PDF_FONT_SIZE = 43,
    SAPI_ENUM_CONF_ID_PDF_ATTRIBUTION = 44,
    SAPI_ENUM_CONF_ID_CERT_VERIFY = 45,
    SAPI_ENUM_CONF_ID_GR_IMG_REDUCE = 46,
    SAPI_ENUM_CONF_ID_MIN_ADES_VERIFY = 47,
    SAPI_ENUM_CONF_ID_INFP_SP_USER_NAME = 48,
    SAPI_ENUM_CONF_ID_INFP_SP_USER_PWD = 49,
    SAPI_ENUM_CONF_ID_INFP_SP_DOMAIN = 50,
    SAPI_ENUM_CONF_ID_INFP_TEMPLATE_NAME = 51,
    SAPI_ENUM_CONF_ID_APPEARANCE_MASK = 52,
    SAPI_ENUM_CONF_ID_OLD_STYLE_PDF_APP = 53,
    SAPI_ENUM_CONF_ID_PDF_FORCE_DISPLAY_SIGNER = 54,
    SAPI_ENUM_CONF_ID_GR_IMG_REDUCE_SCALE = 55,
```

```

SAPI_ENUM_CONF_ID_SIGNATURE_ADDITIONAL_BYTES = 56,
SAPI_ENUM_CONF_ID_XML_ALT_ID                = 57,
SAPI_ENUM_CONF_ID_PADES_ENABLE              = 58,
SAPI_ENUM_CONF_ID_PDF_LOCATOR_OPEN_PATTERN = 59,
SAPI_ENUM_CONF_ID_PDF_LOCATOR_CLOSE_PATTERN = 60,
SAPI_ENUM_CONF_ID_AUTO_GR_IMAGE_DISABLE    = 61,
SAPI_ENUM_CONF_ID_EMBED_CRL_CHAIN           = 62,
SAPI_ENUM_CONF_ID_MAX_CRL_SIZE              = 63,
SAPI_ENUM_CONF_ID_NEXT_CRL_UPDATE_THRESHOLD = 64,
SAPI_ENUM_CONF_ID_THIS_CRL_UPDATE_THRESHOLD = 65,
SAPI_ENUM_CONF_ID_CRITICAL_CRL_ERRORS      = 66,
SAPI_ENUM_CONF_ID_CRL_COMM_TIMEOUT         = 67,
SAPI_ENUM_CONF_ID_OCSP_CACHE_LENGTH        = 68,
} SAPI_ENUM_CONF_ID;

```

### **ENUM values**

**SAPI\_ENUM\_CONF\_ID\_NONE**

Not in use.

**SAPI\_ENUM\_CONF\_ID\_REASON**

**General:** The reason for signing when signing files. This reason is stored in the file and displayed according to the signature field settings.

**Type:** SAPI\_ENUM\_DATA\_TYPE\_STR or SAPI\_ENUM\_DATA\_TYPE\_WSTR.

**Associated Functions:** [SAPISignatureFieldSignEx](#), [SAPISignatureFieldCreateSign](#).

**SAPI\_ENUM\_CONF\_ID\_SEL\_CERT\_TITLE**

**General:** The title of the select certificate dialog box when [SAPICertificateGUISelect](#) is called.

**Type:** SAPI\_ENUM\_DATA\_TYPE\_STR or SAPI\_ENUM\_DATA\_TYPE\_WSTR.

**Associated Functions:** [SAPICertificateGUISelect](#).

**SAPI\_ENUM\_CONF\_ID\_SEL\_CERT\_CMD**

**General:** The command string in the select certificate dialog box when [SAPICertificateGUISelect](#) is called.

**Type:** SAPI\_ENUM\_DATA\_TYPE\_STR or SAPI\_ENUM\_DATA\_TYPE\_WSTR.

**Associated Functions:** [SAPICertificateGUISelect](#).

**SAPI\_ENUM\_CONF\_ID\_CHK\_CRL\_ENUM**

**General:** Whether to filter out certificates whose CRL was not verified correctly during certificates enumeration.

**Type:** SAPI\_ENUM\_DATA\_TYPE\_DWORD.

**Possible Values:** 0, 1.

**Associated Functions:** [SAPICertificatesEnumCont](#).

**SAPI\_ENUM\_CONF\_ID\_CHK\_CRL\_VERIFY**

**General:** Whether to check the CRL while verifying a signature.

**Type:** SAPI\_ENUM\_DATA\_TYPE\_DWORD.

**Possible Values:** 0, 1.

**Associated Functions:** [SAPISignatureFieldVerify](#), [SAPIBufferVerifySignature](#), [SAPIBufferVerifySignatureInit](#), [SAPIBufferVerifySignatureCont](#), [SAPIBufferVerifySignatureEnd](#).

*SAPI\_ENUM\_CONF\_ID\_VERIFY\_CERT\_SIGN*

Not Used.

*SAPI\_ENUM\_CONF\_ID\_PROGRESS\_CALLBACK*

**General:** The pointer of the function receiving the progress events when signing and verifying signature fields in a file. For more information, contact ARX.

*SAPI\_ENUM\_CONF\_ID\_WORD\_SF\_FUNC*

This value is for internal use.

*SAPI\_ENUM\_CONF\_ID\_REASON\_LABEL*

**General:** The label for the reason in a signature field. Applies to PDF files only.

**Type:** SAPI\_ENUM\_DATA\_TYPE\_STR or SAPI\_ENUM\_DATA\_TYPE\_WSTR.

**Associated Functions:** [SAPISignatureFieldCreate](#), [SAPISignatureFieldCreateSign](#).

*SAPI\_ENUM\_CONF\_ID\_DATE\_LABEL*

**General:** The label for the date in a signature field. Applies to PDF files only.

**Type:** SAPI\_ENUM\_DATA\_TYPE\_STR or SAPI\_ENUM\_DATA\_TYPE\_WSTR.

**Associated Functions:** [SAPISignatureFieldCreate](#), [SAPISignatureFieldCreateSign](#).

*SAPI\_ENUM\_CONF\_ID\_SIGNER\_LABEL*

**General:** The label for the signer's name in a signature field. Applies to PDF files only.

**Type:** SAPI\_ENUM\_DATA\_TYPE\_STR or SAPI\_ENUM\_DATA\_TYPE\_WSTR.

**Associated Functions:** [SAPISignatureFieldCreate](#), [SAPISignatureFieldCreateSign](#).

*SAPI\_ENUM\_CONF\_ID\_CERT\_SERIAL\_ID*

**General:** The serial ID of the certificate automatically chosen during signing.

**Type:** SAPI\_ENUM\_DATA\_TYPE\_STR or SAPI\_ENUM\_DATA\_TYPE\_WSTR

**Associated Functions:** [SAPISignatureFieldSignEx](#), [SAPISignatureFieldCreateSign](#).

*SAPI\_ENUM\_CONF\_ID\_CERT\_CHAIN\_FLAGS*

**General:** The flags used for verifying a certificate chain. For exact values, see Microsoft's documentation on [CertGetCertificateChain](#). It is recommended to contact ARX for questions regarding usage of this configuration value.

**Type:** SAPI\_ENUM\_DATA\_TYPE\_DWORD.

**Associated Functions:** All signature generation and signature validation functions.

*SAPI\_ENUM\_CONF\_ID\_PREFERRED\_TOKEN*

**General:** The token to which the graphical image is written by [SAPIGraphicSigImageGet](#), or the token which is used by [SAPIGraphicSigImageGet](#), in an environment of multiple tokens.

Available ARX token names are:

“AR CoSign Appliance for CoSign” – Represents the appliance.

“Token1” – Represents a software token.

“Algorithmic Research Minikey 0” – Represents a MiniKey 6 USB token.

“AKS Ifdh 0” – Represents a MiniKey 7 USB token.

**Type:** SAPI\_ENUM\_DATA\_TYPE\_STR.

**Associated Functions:** [SAPIGraphicSigImageInfoCreate](#), [SAPISigningCeremonyGUI](#).

#### *SAPI\_ENUM\_CONF\_ID\_CERT\_SKI*

**General:** The Subject Key Identifier (based on the public key value) of the certificate automatically chosen during signing.

#### *SAPI\_ENUM\_CONF\_ID\_CERT\_SET\_DEFAULT*

**General:** Whether to set the last certificate used as the default for subsequent signing operations, or not.

**Type:** SAPI\_ENUM\_DATA\_TYPE\_STR or SAPI\_ENUM\_DATA\_TYPE\_WSTR, SAPI\_ENUM\_DATA\_TYPE\_UCHAR.

**Associated Functions:** [SAPISignatureFieldSignEx](#), [SAPISignatureFieldCreateSign](#).

#### *SAPI\_ENUM\_CONF\_ID\_GR\_SIG\_PREF\_NAME*

**General:** The name of the graphical image automatically chosen during signing.

**Type:** SAPI\_ENUM\_DATA\_TYPE\_STR or SAPI\_ENUM\_DATA\_TYPE\_WSTR.

**Associated Functions:** [SAPISignatureFieldSignEx](#), [SAPISignatureFieldCreateSign](#).

#### *SAPI\_ENUM\_CONF\_ID\_PDF\_OWNER\_PWD*

**General:** You can configure a PDF document to have an owner password.

An owner password may be needed to restrict certain operations in the PDF document.

This parameter can be relevant whether a user password is configured or not.

If both a user password and owner password are set in the PDF, it is sufficient to supply an owner password in order to be able to sign the PDF document.

If the PDF is protected only by a user password, it is sufficient to set this configuration parameter in order to be able to sign the PDF document.

**Type:** SAPI\_ENUM\_DATA\_TYPE\_STR or SAPI\_ENUM\_DATA\_TYPE\_WSTR.

**Associated Functions:** [SAPISignatureFieldSignEx](#), [SAPISignatureFieldCreateSign](#).

#### *SAPI\_ENUM\_CONF\_ID\_PDF\_USER\_PWD*

**General:** A PDF document can be configured to have a user password.

This parameter contains the user password needed to open a protected PDF document.

**Type:** SAPI\_ENUM\_DATA\_TYPE\_STR or SAPI\_ENUM\_DATA\_TYPE\_WSTR.

**Associated Functions:** [SAPISignatureFieldSignEx](#), [SAPISignatureFieldCreateSign](#).

#### *SAPI\_ENUM\_CONF\_ID\_PDF\_SF\_FUNC*

**General:** This value is for internal use.

#### *SAPI\_ENUM\_CONF\_ID\_GMT\_OFFSET*

**General:** This value is for setting a GMT offset that is different from the current system settings. The value is in minutes.

**Type:** SAPI\_ENUM\_DATA\_TYPE\_DWORD.

**Associated Functions:** [SAPISignatureFieldSignEx](#), [SAPISignatureFieldCreateSign](#).

*SAPI\_ENUM\_CONF\_ID\_RET\_EXT\_FIELD\_INFO*

**General:** This value is used to notify the [SAPISignatureFieldInfoGet](#) function to retrieve information that is kept inside custom fields. Set the configuration value prior to calling [SAPISignatureFieldInfoGet](#).

**Type:** SAPI\_ENUM\_DATA\_TYPE\_DWORD.

**Possible Values:** 0, 1.

**Associated Functions:** [SAPISignatureFieldInfoGet](#).

*SAPI\_ENUM\_CONF\_ID\_SECURED\_TS\_ENABLE*

**General:** Whether to enable or disable the time-stamping service. For more information see [Time-Stamping/OCSP and Miscellaneous Signature-Related Flags](#).

**Type:** SAPI\_ENUM\_DATA\_TYPE\_DWORD.

**Possible Values:** 0, 1.

**Associated Functions:** [SAPISignatureFieldSignEx](#), [SAPISignatureFieldCreateSign](#), [SAPIBufferSignEx](#), [SAPIBufferSignInit](#), [SAPIBufferSignCont](#), [SAPIBufferSignEnd](#).

*SAPI\_ENUM\_CONF\_ID\_SECURED\_TS\_URL*

**General:** The URL of the time-stamping service.

**Type:** SAPI\_ENUM\_DATA\_TYPE\_STR, SAPI\_ENUM\_DATA\_TYPE\_WSTR.

**Associated Functions:** [SAPISignatureFieldSignEx](#), [SAPISignatureFieldCreateSign](#), [SAPIBufferSignEx](#), [SAPIBufferSignInit](#), [SAPIBufferSignCont](#), [SAPIBufferSignEnd](#).

*SAPI\_ENUM\_CONF\_ID\_SECURED\_TS\_USER*

**General:** The user-ID used to authenticate to the time-stamping service.

**Type:** SAPI\_ENUM\_DATA\_TYPE\_STR, SAPI\_ENUM\_DATA\_TYPE\_WSTR.

**Associated Functions:** [SAPISignatureFieldSignEx](#), [SAPISignatureFieldCreateSign](#), [SAPIBufferSignEx](#), [SAPIBufferSignInit](#), [SAPIBufferSignCont](#), [SAPIBufferSignEnd](#).

*SAPI\_ENUM\_CONF\_ID\_SECURED\_TS\_PWD*

**General:** The password of the user-ID used to authenticate to the time-stamping service.

**Type:** SAPI\_ENUM\_DATA\_TYPE\_STR, SAPI\_ENUM\_DATA\_TYPE\_WSTR.

**Associated Functions:** [SAPISignatureFieldSignEx](#), [SAPISignatureFieldCreateSign](#), [SAPIBufferSignEx](#), [SAPIBufferSignInit](#), [SAPIBufferSignCont](#), [SAPIBufferSignEnd](#).

*SAPI\_ENUM\_CONF\_ID\_SECURED\_TS\_ADDITIONAL\_BYTES*

**General:** The amount of bytes to be added to the PKCS#7 blob to maintain the time-stamping object. This parameter is necessary when signing a PDF document using a time-stamping service. Its value must be more than the expected size of the time-stamping blob.

**Type:** SAPI\_ENUM\_DATA\_TYPE\_DWORD.

**Associated Functions:** [SAPISignatureFieldSignEx](#), [SAPISignatureFieldCreateSign](#), [SAPIBufferSignEx](#), [SAPIBufferSignInit](#), [SAPIBufferSignCont](#), [SAPIBufferSignEnd](#).

#### *SAPI\_ENUM\_CONF\_ID\_SECURED\_TS\_CERT\_STATUS*

**General:** Provides the exact time-stamping certificate status. This configuration value should be retrieved by the application after the signature verification operation.

**Associated Functions:** [SAPISignatureFieldVerify](#), [SAPIBufferVerifySignature](#), [SAPIBufferVerifySignatureInit](#), [SAPIBufferVerifySignatureCont](#), [SAPIBufferVerifySignatureEnd](#).

#### *SAPI\_ENUM\_CONF\_ID\_TITLE*

**General:** The current title to be used as part of the signature operation.

**Type:** SAPI\_ENUM\_DATA\_TYPE\_STR or SAPI\_ENUM\_DATA\_TYPE\_WSTR.

**Associated Functions:** [SAPISignatureFieldSignEx](#), [SAPISignatureFieldCreateSign](#).

#### *SAPI\_ENUM\_CONF\_ADDITIONAL\_TEXT*

**General:** This configuration value is identical to the SAPI\_ENUM\_CONF\_ID\_TITLE enumerand.

**Type:** SAPI\_ENUM\_DATA\_TYPE\_STR or SAPI\_ENUM\_DATA\_TYPE\_WSTR.

**Associated Functions:** [SAPISignatureFieldSignEx](#), [SAPISignatureFieldCreateSign](#).

#### *SAPI\_ENUM\_CONF\_ID\_PDF\_ROAMING\_ID\_SRV*

**General:** Set a roaming ID server information. This information is relevant to the Adobe/PDF roaming ID profile.

This string value must not be an empty string.

**Type:** SAPI\_ENUM\_DATA\_TYPE\_STR or SAPI\_ENUM\_DATA\_TYPE\_WSTR.

**Associated Functions:** [SAPISignatureFieldCreate](#), [SAPISignatureFieldCreateSign](#).

#### *SAPI\_ENUM\_CONF\_ID\_KEY\_CONTAINER\_NAME*

**General:** This parameter contains a CAPI virtual reader name that symbolizes the PKCS#11 slot that is allocated during a SAPILogonAPI call. The application can retrieve this value after the SAPILogon call.

**Type:** SAPI\_ENUM\_DATA\_TYPE\_STR.

**Associated Functions:** [SAPILogonEx](#).

#### *SAPI\_ENUM\_CONF\_ID\_SLOT\_NUMBER*

**General:** This parameter contains a PKCS#11 slot number, when SAPILogon API is used. The application can retrieve this value after the SAPILogon call.

**Type:** SAPI\_ENUM\_DATA\_TYPE\_DWORD.

**Associated Functions:** [SAPILogonEx](#).

#### *SAPI\_ENUM\_CONF\_ID\_LOGO\_PREF\_NAME*

**General:** The name of the logo automatically chosen during signing.

**Type:** SAPI\_ENUM\_DATA\_TYPE\_STR or SAPI\_ENUM\_DATA\_TYPE\_WSTR.

**Associated Functions:** [SAPISignatureFieldSignEx](#), [SAPISignatureFieldCreateSign](#).

#### *SAPI\_ENUM\_CONF\_ID\_LOCK\_USER*

**General:** Set this configuration value to 1 if a third-party application is used as part of a user session to sign data or a document, and the application uses the Microsoft certificate store to select the user certificate. The

configuration value should be set prior to the SAPILogon API Call. After the SAPILogin, call any attempt to view the certificate store will show the user related certificates.

**Type:** SAPI\_ENUM\_DATA\_TYPE\_DWORD.

**Possible Values:** 0, 1.

**Associated Functions:** [SAPILogonEx](#).

*SAPI\_ENUM\_CONF\_ID\_EMBED\_OCSP*

**General:** Set this configuration value to 1 to include an OCSP response in the signature blob. This will enable a verifier to make sure that the signer certificate and all other certificates in the chain were valid at the time of the signature.

The embedding of the OCSP response is compatible with Adobe PDF definitions.

**Type:** SAPI\_ENUM\_DATA\_TYPE\_DWORD.

**Possible Values:** 0,1.

**Associated Functions:** [SAPISignatureFieldSignEx](#), [SAPISignatureFieldCreateSign](#), [SAPIBufferSignEx](#), [SAPIBufferSignInit](#), [SAPIBufferSignCont](#), [SAPIBufferSignEnd](#).

*SAPI\_ENUM\_CONF\_ID\_OCSP\_URL*

**General:** If it is required to embed an OCSP response into the digital signature, use this value to define the URL of the OCSP server. This value is used in the case that no OCSP URL is defined in the relevant certificate (either the signer certificate or any other certificate in the certificate chain).

**Type:** ENUM\_DATA\_TYPE\_STR or SAPI\_ENUM\_DATA\_TYPE\_WSTR.

**Associated Functions:** [SAPISignatureFieldSignEx](#), [SAPISignatureFieldCreateSign](#), [SAPIBufferSignEx](#), [SAPIBufferSignInit](#), [SAPIBufferSignCont](#), [SAPIBufferSignEnd](#).

*SAPI\_ENUM\_CONF\_ID\_CERT\_PUBLIC*

**General:** The public key of the certificate automatically chosen during signing. The value should be passed in hexadecimal string format.

**Type:** ENUM\_DATA\_TYPE\_STR or SAPI\_ENUM\_DATA\_TYPE\_WSTR.

**Associated Functions:** [SAPISignatureFieldSignEx](#), [SAPISignatureFieldCreateSign](#), [SAPIBufferSignEx](#), [SAPIBufferSignInit](#), [SAPIBufferSignCont](#), [SAPIBufferSignEnd](#).

*SAPI\_ENUM\_CONF\_ID\_TIFF\_BANNER\_CREATE*

**General:** A visible signature can be embedded into a TIFF document. This replaces the previous implementation that added a banner page at the beginning of the TIFF document.

If you set this configuration value to 1, the previous functionality is preserved.

**Type:** SAPI\_ENUM\_DATA\_TYPE\_DWORD.

**Possible Values :**0, 1.

**Associated Functions:** [SAPISignatureFieldCreate](#).

*SAPI\_ENUM\_CONF\_ID\_TIFF\_IGNORE\_EMBEDDED\_CLEAR*

**General:** This configuration value is used during the creation of a new signature field. It directs the TIFF provider of CoSign Signature Local to disallow a clear operation when using a visible signature field.

**Type:** SAPI\_ENUM\_DATA\_TYPE\_DWORD

**Possible Values:** 0, 1.

**Associated Functions:** [SAPISignatureFieldCreate](#).

*SAPI\_ENUM\_CONF\_ID\_XADES\_TEMPLATE*

**General:** This configuration value enables using XML data that can be incorporated into the signed object as part of an advanced XML signature operation. This option enables XAdES-PES support. For more information, contact ARX. When used in CoSign Signature SOAP, this value should be passed encoded in base64 format.

**Type:** ENUM\_DATA\_TYPE\_STR or SAPI\_ENUM\_DATA\_TYPE\_WSTR.

**Associated Functions:** [SAPISignatureFieldCreateSign](#).

*SAPI\_ENUM\_CONF\_ID\_EXTENDED\_VALIDATION*

**General:** Specifies that an extended certificate validation path will be used before attempting to sign. This meets the DoD certificate path validation requirements.

**Type:** SAPI\_ENUM\_DATA\_TYPE\_DWORD.

**Possible Values:**0, 1.

**Associated Functions:** [SAPICertificatesEnumInit](#), [SAPICertificatesEnumCont](#), [SAPISignatureFieldSignEx](#), [SAPISignatureFieldCreateSign](#), [SAPIBufferSignEx](#), [SAPIBufferSignInit](#), [SAPIBufferSignCont](#), [SAPIBufferSignEnd](#).

*SAPI\_ENUM\_CONF\_ID\_PDF\_FONT\_PATH*

Reserved for future implementations of CoSign Signature Local.

*SAPI\_ENUM\_CONF\_ID\_PDF\_FONT\_PATH*

Reserved for future implementations of CoSign Signature Local.

*SAPI\_ENUM\_CONF\_ID\_PDF\_FONT\_SIZE*

Reserved for future implementations of CoSign Signature Local.

*SAPI\_ENUM\_CONF\_ID\_PDF\_ATTRIBUTION*

**General:** Specifies whether to embed the attribution in PDF signatures.

**Type:** SAPI\_ENUM\_DATA\_TYPE\_DWORD

**Possible Values:** 0, 1.

**Associated Functions:** [SAPISignatureFieldSignEx](#), [SAPISignatureFieldCreateSign](#).

*SAPI\_ENUM\_CONF\_ID\_CERT\_VERIFY*

**General:** In the case of PKCS#1 signature validation, use this configuration value to include the certificate value to be used for signature validation. The value of the certificate should be DER encoded as a byte array.

**Type:** SAPI\_ENUM\_DATA\_TYPE\_UCHAR.

**Associated Functions:** [SAPIBufferSignEx](#), [SAPIBufferSignInit](#), [SAPIBufferSignCont](#), [SAPIBufferSignEnd](#).

#### *SAPI\_ENUM\_CONF\_ID\_IMAGE\_REDUCE*

**General:** Activate the image size reduction functionality when uploading new graphical signatures into DocuSign Signature Appliance.

To use the default image size reduction, pass the value of `PERFORM_IMAGE_REDUCTION_DEFAULT`. If you do not want any image size reduction, pass the value of `NOT_PERFORM_IMAGE_REDUCTION`.

**Type:** `SAPI_ENUM_DATA_TYPE_DWORD`.

**Possible Values:** `NOT_PERFORM_IMAGE_REDUCTION`, `PERFORM_IMAGE_REDUCTION_DEFAULT`.

**Associated Functions:** [SAPIGraphicSigImageInfoCreate](#), [SAPIGraphicSigImageInfoUpdate](#).

#### *SAPI\_ENUM\_CONF\_ID\_MIN\_ADES\_VERIFY*

**General:** Use this configuration value to enforce a minimal level of existing attributes, such as signature time or signer certificate, in the signature. The minimum level is defined according to the [SAPI\\_ENUM\\_ADVANCED\\_SIG\\_LEVEL](#) enumeration type.

**Type:** `SAPI_ENUM_DATA_TYPE_DWORD`.

**Possible Values:** refer to `SAPI_ENUM_ADVANCED_SIG_LEVEL`.

**Associated Functions:** [SAPISignatureFieldVerify](#), [SAPIBufferVerifySignature](#), [SAPIBufferVerifySignatureInit](#), [SAPIBufferVerifySignatureCont](#), [SAPIBufferVerifySignatureEnd](#).

#### *SAPI\_ENUM\_CONF\_ID\_INFP\_SP\_USER\_NAME*

**General:** Specify the required Microsoft SharePoint user name when accessing SharePoint's InfoPath files and templates.

**Type:** `SAPI_ENUM_DATA_TYPE_STR` or `SAPI_ENUM_DATA_TYPE_WSTR`.

**Associated Functions:** [SAPISignatureFieldSignEx](#).

#### *SAPI\_ENUM\_CONF\_ID\_INFP\_SP\_USER\_PWD*

**General:** Specify the required Microsoft SharePoint user's password when accessing SharePoint's InfoPath files and templates.

**Type:** `SAPI_ENUM_DATA_TYPE_STR` or `SAPI_ENUM_DATA_TYPE_WSTR`.

**Associated Functions:** [SAPISignatureFieldSignEx](#).

#### *SAPI\_ENUM\_CONF\_ID\_INFP\_SP\_DOMAIN*

**General:** Specify the required Microsoft SharePoint user's domain name when accessing SharePoint's InfoPath files and templates.

**Type:** `SAPI_ENUM_DATA_TYPE_STR` or `SAPI_ENUM_DATA_TYPE_WSTR`.

**Associated Functions:** [SAPISignatureFieldSignEx](#).

#### *SAPI\_ENUM\_CONF\_ID\_INFP\_TEMPLATE\_NAME*

**General:** Provide a URL or local file location for accessing the InfoPath template file. In the case of a local file, provide a file location such as the following: `file:///dir1/dir2/<file Name>`.

**Type:** `SAPI_ENUM_DATA_TYPE_STR` or `SAPI_ENUM_DATA_TYPE_WSTR`.

**Associated Functions:** [SAPISignatureFieldSignEx](#).

### *SAPI\_ENUM\_CONF\_ID\_APPEARANCE\_MASK*

**General:** In cases where CoSign Signature Local does not have control over signature fields such as in InfoPath, you can pass an appearance mask as part of the digital signature operation. An appearance mask is a bit mask indicating what information is displayed when the signature field is signed, such as graphical image, signer, reason, date, etc. (for the mask values description, refer to [SAPI\\_ENUM\\_DRAWING\\_ELEMENT](#)). If you pass an appearance mask in cases where CoSign Signature Local does not have control over signature fields, the visible signature content is displayed according to the appearance mask, but the appearance mask attribute is not kept in the signature field.

**Type:** SAPI\_ENUM\_DATA\_TYPE\_DWORD.

**Associated Functions:** [SAPISignatureFieldSignEx](#).

### *SAPI\_ENUM\_CONF\_ID\_OLD\_STYLE\_PDF\_APP*

**General:** Whether to use the new method (introduced in CoSign version 5.6.4) of representing the visible signature in a PDF file. In the old method, each component of the visible signature – each text element and each graphical element – was saved separately. In the new method, all the elements are combined into a single image that is incorporated into the PDF document. The new method is more suitable for PDF/A requirements, and provides a solution for dealing with Unicode languages such as Chinese and Japanese. A value of 1 indicates the previous visible signature format is used.

**Type:** SAPI\_ENUM\_DATA\_TYPE\_DWORD.

**Possible Values:** 0 (new method – the default), 1 (old method).

**Associated Functions:** [SAPISignatureFieldSignEx](#), [SAPISignatureFieldCreateSign](#).

### *SAPI\_ENUM\_CONF\_ID\_PDF\_FORCE\_DISPLAY\_SIGNER*

**General:** Relevant only to PDF. When this parameter is set to 1, the signer's name will always appear.

**Type:** SAPI\_ENUM\_DATA\_TYPE\_DWORD.

**Associated Functions:** [SAPISignatureFieldSignEx](#).

### *SAPI\_ENUM\_CONF\_ID\_GR\_IMG\_REDUCE\_SCALE*

**General:** Directs CoSign Signature Local to reduce the size of an uploaded graphical image if it exceeds the max size limit (30 K).

**Type:** SAPI\_ENUM\_DATA\_TYPE\_DWORD.

**Associated Functions:** [SAPIGraphicSigImageInfoCreate](#) and [SAPIGraphicSigImageInfoUpdate](#).

### *SAPI\_ENUM\_CONF\_ID\_SIGNATURE\_ADDITIONAL\_BYTES*

**General:** The number of bytes to be added to the PKCS#7 blob to maintain additional information such as OCSP data.

**Type:** SAPI\_ENUM\_DATA\_TYPE\_DWORD.

**Associated Functions:** [SAPISignatureFieldSignEx](#).

### *SAPI\_ENUM\_CONF\_ID\_XML\_ALT\_ID*

**General:** An alternative name for the ID attribute that represents the data being signed in an signed XML structure.

**Type:** SAPI\_ENUM\_DATA\_TYPE\_STR.

**Associated Functions:** [SAPISignatureFieldSignEx](#).

#### *SAPI\_ENUM\_CONF\_ID\_PADES\_ENABLE*

**General:** Whether to generate a PAdES-compliant PDF signature. In a PAdES compliant signature, the identification of the digital signature is *ETSI.CAdES.detached*, whereas in a regular PDF signature it is *adbe.pkcs7.detached*. This is supported from Adobe 10 onwards.

**Type:** SAPI\_ENUM\_DATA\_TYPE\_DWORD.

**Associated Functions:** [SAPISignatureFieldSignEx](#).

#### *SAPI\_ENUM\_CONF\_ID\_PDF\_LOCATOR\_OPEN\_PATTERN*

**General:** Enable setting the PDF field locator opening pattern. The opening pattern will be specified as a string value.

**Type:** SAPI\_ENUM\_DATA\_TYPE\_STR.

**Associated Functions:** [SAPISignatureFieldSignEx](#).

#### *SAPI\_ENUM\_CONF\_ID\_PDF\_LOCATOR\_CLOSE\_PATTERN*

**General:** Enable setting PDF field locator closing pattern. The closing pattern will be specified as a string value.

**Type:** SAPI\_ENUM\_DATA\_TYPE\_STR.

**Associated Functions:** [SAPISignatureFieldSignEx](#).

#### *SAPI\_ENUM\_CONF\_ID\_AUTO\_GR\_IMAGE\_DISABLE*

**General:** Disable generating automatic graphical signatures in the session. The automated signature is generated in the session if the user has not yet defined his/her graphical signatures.

**Type:** SAPI\_ENUM\_DATA\_TYPE\_DWORD.

**Possible Values:** 0 (Auto graphical image is enabled), 1 (Auto graphical image is disabled).

**Associated Functions:** [SAPIGraphicSigImageEnumInit](#)

#### *SAPI\_ENUM\_CONF\_ID\_EMBED\_CRL\_CHAIN*

**General:** Whether to embed a CRL inside the PKCS#7 signature. This is relevant to PDF signatures, Buffer signatures, Legacy Word add-in and Tiff signatures.

**Type:** SAPI\_ENUM\_DATA\_TYPE\_DWORD.

**Possible Values:** 0 (No CRL is embedded into the signature), 1 (CRL is embedded into the signature).

**Associated Functions:** [SAPISignatureFieldSignEx](#).

#### *SAPI\_ENUM\_CONF\_ID\_MAX\_CRL\_SIZE*

**General:** The maximum CRL size to embed into the signature. If the CRL is larger, the operation fails.

**Type:** SAPI\_ENUM\_DATA\_TYPE\_DWORD.

**Associated Functions:** [SAPISignatureFieldSignEx](#).

#### *SAPI\_ENUM\_CONF\_ID\_NEXT\_CRL\_UPDATE\_THRESHOLD*

**General:** Minimal time gap (in seconds) between the current time and CRL next update time. This parameter is used for caching purposes.

If the time left until the next CRL update is less than the Threshold, the local cache is not used.

**Type:** SAPI\_ENUM\_DATA\_TYPE\_DWORD.

**Associated Functions:** [SAPISignatureFieldSignEx](#).

*SAPI\_ENUM\_CONF\_ID\_THIS\_CRL\_UPDATE\_THRESHOLD*

**General:** Maximal time gap (in seconds) between the current update time and the CRL current time. This parameter is used for caching purposes.

If the time from CRL publication is more than the specified threshold, the local cache is not used.

**Type:** SAPI\_ENUM\_DATA\_TYPE\_DWORD.

**Associated Functions:** [SAPISignatureFieldSignEx](#).

*SAPI\_ENUM\_CONF\_ID\_CRITICAL\_CRL\_ERRORS*

**General:** Indicates whether an error in the CRL retrieval should stop the digital signature operation.

**Type:** SAPI\_ENUM\_DATA\_TYPE\_DWORD.

**Possible Values:** 0 (continue with signature operation), 1 (stop the signature operation).

**Associated Functions:** [SAPISignatureFieldSignEx](#).

*SAPI\_ENUM\_CONF\_ID\_CRL\_COMM\_TIMEOUT*

**General:** Timeout (in milliseconds) for CRL fetching operations. If the timeout expires, the CRL fetching is considered an error.

**Type:** SAPI\_ENUM\_DATA\_TYPE\_DWORD.

**Associated Functions:** [SAPISignatureFieldSignEx](#).

*SAPI\_ENUM\_CONF\_ID\_OCSP\_CACHE\_LENGTH*

**General:** Size in bytes of internal caching of OCSP responses. OCSP responses can be added to the digital signature information for long term validation.

**Type:** SAPI\_ENUM\_DATA\_TYPE\_DWORD.

**Associated Functions:** [SAPISignatureFieldSignEx](#).

## SAPI\_ENUM\_DATA\_TYPE

Data types:

```
typedef enum SAPI_ENUM_DATA_TYPE {
    SAPI_ENUM_DATA_TYPE_NONE           = 0,
    SAPI_ENUM_DATA_TYPE_DWORD          = 1,
    SAPI_ENUM_DATA_TYPE_STR             = 2,
    SAPI_ENUM_DATA_TYPE_WSTR           = 3,
    SAPI_ENUM_DATA_TYPE_TIME            = 4,
    SAPI_ENUM_DATA_TYPE_UCHAR           = 5,
    SAPI_ENUM_DATA_TYPE_FUNC_PTR        = 6,
    SAPI_ENUM_DATA_TYPE_SAPI_FILE_TIME  = 7
} SAPI_ENUM_DATA_TYPE;
```

### **ENUM values**

*SAPI\_ENUM\_DATA\_TYPE\_NONE*

Not in use.

*SAPI\_ENUM\_DATA\_TYPE\_DWORD*

Double word (unsigned long).

*SAPI\_ENUM\_DATA\_TYPE\_STR*

String (char \*).

*SAPI\_ENUM\_DATA\_TYPE\_WSTR*

Wide char string (WCHAR \*).

*SAPI\_ENUM\_DATA\_TYPE\_TIME*

Time (time\_t – long).

*SAPI\_ENUM\_DATA\_TYPE\_UCHAR*

Pointer to unsigned char.

*SAPI\_ENUM\_DATA\_TYPE\_FUNC\_PTR*

Function pointer.

*SAPI\_ENUM\_DATA\_TYPE\_SAPI\_FILE\_TIME*

CoSign Signature Local time definition.

## SAPI\_ENUM\_IMAGE\_SOURCE\_TYPE

The source of the graphical image:

```
typedef enum SAPI_ENUM_IMAGE_SOURCE_TYPE {  
    SAPI_ENUM_IMAGE_SOURCE_NONE      = 0,  
    SAPI_ENUM_IMAGE_SOURCE_PAD        = 1,  
    SAPI_ENUM_IMAGE_SOURCE_TOKEN      = 2  
} SAPI_ENUM_IMAGE_SOURCE_TYPE;
```

### ***ENUM values***

*SAPI\_ENUM\_IMAGE\_SOURCE\_NONE*

Not in use.

*SAPI\_ENUM\_IMAGE\_SOURCE\_PAD*

The image is taken from a pad. Not supported in this version.

*SAPI\_ENUM\_IMAGE\_SOURCE\_TOKEN*

The image is taken from a token.

## SAPI\_ENUM\_CERT\_STATUS

Certificate status as interpreted by CoSign Signature Local.

```
typedef SAPI_ENUM_CERT_STATUS {
    SAPI_ENUM_CERT_STATUS_NOT_CHECKED    = 0,
    SAPI_ENUM_CERT_STATUS_OK             = 1,
    SAPI_ENUM_CERT_STATUS_REVOKED        = 2,
    SAPI_ENUM_CERT_STATUS_INVALID         = 3,
    SAPI_ENUM_CERT_STATUS_WARNING         = 4,
    SAPI_ENUM_TS_CERT_STATUS_REVOKED      = 5,
    SAPI_ENUM_TS_CERT_STATUS_INVALID      = 6
} SAPI_ENUM_CERT_STATUS;
```

### **ENUM values**

*SAPI\_ENUM\_CERT\_STATUS\_NOT\_CHECKED*

The certificate status was not checked.

*SAPI\_ENUM\_CERT\_STATUS\_OK*

The certificate is OK.

*SAPI\_ENUM\_CERT\_STATUS\_REVOKED*

The certificate is revoked.

*SAPI\_ENUM\_CERT\_STATUS\_INVALID*

The certificate is invalid. The signature might not be trusted.

*SAPI\_ENUM\_CERT\_STATUS\_WARNING*

Not in use.

*SAPI\_ENUM\_TS\_CERT\_STATUS\_REVOKED*

Whether the certificate of the time-stamping server is revoked.

*SAPI\_ENUM\_TS\_CERT\_STATUS\_INVALID*

Whether there is a warning to validate the time-stamping server certificate.

## SAPI\_ENUM\_CERT\_FIELD

The values that can be retrieved from a certificate.

```
typedef SAPI_ENUM_CERT_FIELD {
    SAPI_ENUM_CERT_FIELD_NONE           = 0,
    SAPI_ENUM_CERT_FIELD_CERT           = 1,
    SAPI_ENUM_CERT_FIELD_SUBJECT        = 2,
    SAPI_ENUM_CERT_FIELD_ISSUER         = 3,
    SAPI_ENUM_CERT_FIELD_EMAIL          = 4,
    SAPI_ENUM_CERT_FIELD_NOT_BEFORE     = 5,
    SAPI_ENUM_CERT_FIELD_NOT_AFTER      = 6,
    SAPI_ENUM_CERT_FIELD_SERIAL_ID      = 7,
    SAPI_ENUM_CERT_FIELD_SKI            = 8,
    SAPI_ENUM_CERT_FIELD_HASHED_ISSUER_NAME = 9,
    SAPI_ENUM_CERT_FIELD_HASHED_ISSUER_KEY = 10,
    SAPI_ENUM_CERT_FIELD_OCSP_URL       = 11,
    SAPI_ENUM_CERT_FIELD_PUBLIC_KEY     = 12,
} SAPI_ENUM_CERT_FIELD;
```

### *ENUM values*

*SAPI\_ENUM\_CERT\_FIELD\_NONE*

Not in use.

*SAPI\_ENUM\_CERT\_FIELD\_CERT*

The ASN1 encoded certificate value.

*SAPI\_ENUM\_CERT\_FIELD\_SUBJECT*

The certificate subject (the owner of the certificate).

*SAPI\_ENUM\_CERT\_FIELD\_ISSUER*

The issuer of the certificate.

*SAPI\_ENUM\_CERT\_FIELD\_EMAIL*

The email of the certificate's owner.

*SAPI\_ENUM\_CERT\_FIELD\_NOT\_BEFORE*

The time from when the certificate is valid.

*SAPI\_ENUM\_CERT\_FIELD\_NOT\_AFTER*

The expiration time of the certificate.

*SAPI\_ENUM\_CERT\_FIELD\_SERIAL\_ID*

The serial ID of the certificate.

*SAPI\_ENUM\_CERT\_FIELD\_SKI*

The Subject Key Identifier (based on the public key value) of the certificate.

*SAPI\_ENUM\_CERT\_FIELD\_SKI*

The Subject Key Identifier (based on the public key value) of the certificate.

*SAPI\_ENUM\_CERT\_FIELD\_HASHED\_ISSUER\_NAME*

A hash value of the issuer field in the certificate.

*SAPI\_ENUM\_CERT\_FIELD\_HASHED\_ISSUER\_KEY*

A hash value of the issuer's public key information.

*SAPI\_ENUM\_CERT\_FIELD\_FIELD\_OCSP\_URL*

The URL of an OCSP server that can retrieve the OCSP response.

*SAPI\_ENUM\_CERT\_FIELD\_PUBLIC\_KEY*

The user's public key value.

## SAPI\_ENUM\_PKCS7\_FIELD

The values that can be retrieved from a PKCS#7 blob.

```
typedef SAPI_ENUM_PKCS7_FIELD {
    SAPI_ENUM_PKCS7_FIELD_NONE           = 0,
    SAPI_ENUM_PKCS7_FIELD_CERT           = 1,
    SAPI_ENUM_PKCS7_FIELD_SIGNATURE      = 2,
    SAPI_ENUM_PKCS7_FIELD_TIME           = 3,
    SAPI_ENUM_PKCS7_FIELD_HASH           = 4,
    SAPI_ENUM_PKCS7_FIELD_HASH_ALG       = 5,
    SAPI_ENUM_PKCS7_FIELD_SIGNED_CONTENT = 6,
    SAPI_ENUM_PKCS7_FIELD_SIGNED_CERT    = 7
} SAPI_ENUM_PKCS7_FIELD;
```

### *ENUM values*

#### *SAPI\_ENUM\_PKCS7\_FIELD\_NONE*

Not in use.

#### *SAPI\_ENUM\_PKCS7\_FIELD\_CERT*

The ASN1 encoded certificate value.

#### *SAPI\_ENUM\_PKCS7\_FIELD\_SIGNATURE*

The Signature value inside the PKCS#7 blob.

#### *SAPI\_ENUM\_PKCS7\_FIELD\_TIME*

The time value inside the PKCS#7 blob. If no time-stamping is defined, the regular time inside the PKCS#7 blob is returned.

#### *SAPI\_ENUM\_PKCS7\_FIELD\_HASH*

The hash of the data inside the PKCS#7 blob. This value is located in the signed attributes section. Older signatures do not have this attribute.

#### *SAPI\_ENUM\_PKCS7\_FIELD\_HASH\_ALG*

The signature hash algorithm used to sign the hash value.

#### *SAPI\_ENUM\_PKCS7\_SIGNED\_CONTENT*

The signed attributes section inside the PKCS#7 signature.

#### *SAPI\_ENUM\_PKCS7\_SIGNED\_CERT*

The hash of the signature certificate. This value is embedded in the signed attributes. This is one of the basic criteria of an advanced signature.

## SAPI\_ENUM\_STORE\_TYPE

The types of system store in which a certificate can be installed.

```
typedef SAPI_ENUM_STORE_TYPE {  
    SAPI_ENUM_STORE_NONE           = 0,  
    SAPI_ENUM_STORE_USER           = 1,  
    SAPI_ENUM_STORE_LOCAL_MACHINE = 2  
} SAPI_ENUM_STORE_TYPE;
```

### ***ENUM values***

*SAPI\_ENUM\_STORE\_NONE*

Not in use.

*SAPI\_ENUM\_STORE\_USER*

The user store.

*SAPI\_ENUM\_STORE\_LOCAL\_MACHINE*

The local machine store.

## SAPI\_ENUM\_STRUCT\_TYPE

The structure types whose memory is allocated by CoSign Signature Local and is freed by the [SAPIStructRelease](#) function.

```
typedef SAPI_ENUM_STRUCT_TYPE {
    SAPI_ENUM_STRUCT_TYPE_NONE           = 0,
    SAPI_ENUM_STRUCT_TYPE_SIGNED_FIELD   = 1,
    SAPI_ENUM_STRUCT_TYPE_GR_IMAGE       = 2,
    SAPI_ENUM_STRUCT_TYPE_GR_SIG_INFO     = 3,
    SAPI_ENUM_STRUCT_TYPE_FIELD_INFO      = 4
} SAPI_ENUM_STRUCT_TYPE;
```

### ***ENUM values***

**SAPI\_ENUM\_STRUCT\_TYPE\_NONE**

Not in use.

**SAPI\_ENUM\_STRUCT\_TYPE\_SIGNED\_FIELD**

The signed field information structure ([SAPI\\_SIGNED\\_FIELD\\_INFO](#)).

**SAPI\_ENUM\_STRUCT\_TYPE\_GR\_IMAGE**

The graphical image structure ([SAPI\\_GRAPHIC\\_IMAGE\\_STRUCT](#)).

**SAPI\_ENUM\_STRUCT\_TYPE\_GR\_SIG\_INFO**

The graphical image information structure ([SAPI\\_GR\\_IMG\\_INFO](#)).

**SAPI\_ENUM\_STRUCT\_TYPE\_FIELD\_INFO**

This structure contains the array of custom fields.

## SAPI\_ENUM\_GR\_IMG\_SELECT\_MODE

The different modes in which the graphical image selection GUI dialog can operate.

```
typedef SAPI_ENUM_GR_IMG_SELECT_MODE {  
    SAPI_ENUM_GR_IMG_SEL_MODE_SELECT      = 0,  
    SAPI_ENUM_GR_IMG_SEL_MODE_VIEW_USER   = 1,  
    SAPI_ENUM_GR_IMG_SEL_MODE_VIEW_KIOSK  = 2,  
    SAPI_ENUM_GR_IMG_SEL_MODE_VOLATILE    = 3  
} SAPI_ENUM_GR_IMG_SELECT_MODE;
```

### *ENUM values*

#### *SAPI\_ENUM\_GR\_IMG\_SEL\_MODE\_SELECT*

The graphical image selection dialog box is called for selecting a graphical image. If only one image exists, the dialog box does not appear. It is possible to edit the images in DocuSign Signature Appliance when the dialog box is called in this mode.

#### *SAPI\_ENUM\_GR\_IMG\_SEL\_MODE\_VIEW\_USER*

The graphical image selection dialog box is called for editing the images in DocuSign Signature Appliance and it runs under the logged-in user. Only this mode enables updating the graphical image in Adobe Acrobat/Reader.

#### *SAPI\_ENUM\_GR\_IMG\_SEL\_MODE\_VIEW\_KIOSK*

The graphical image selection dialog box is called for editing the images in DocuSign Signature Appliance and it requires an explicit login by a user. In this mode, it is not possible to update the graphical image in Adobe Acrobat/Reader.

#### *SAPI\_ENUM\_GR\_IMG\_SEL\_MODE\_VOLATILE*

This mode can be used for capturing a graphical image intended for the current transaction only; that is, a temporary graphical signature.

## SAPI\_ENUM\_SIG\_FIELD\_SETTINGS\_ID

The types of information that can be kept inside a signature field.

```
typedef SAPI_ENUM_SIG_FIELD_SETTINGS_ID {
    SAPI_ENUM_SIG_FIELD_SETTING_NONE           = 0,
    SAPI_ENUM_SIG_FIELD_SETTING_CLEAR_PERMIT   = 1,
    SAPI_ENUM_SIG_FIELD_SETTING_TITLE          = 2,
    SAPI_ENUM_SIG_FIELD_SETTING_TITLE_TAG      = 3,
    SAPI_ENUM_SIG_FIELD_SETTING_SUGGESTED_SIGNER = 4,
    SAPI_ENUM_SIG_FIELD_SETTING_INSTRUCT_TO_SIGNER = 5,
    SAPI_ENUM_SIG_FIELD_SETTING_ALLOW_REASON   = 6,
    SAPI_ENUM_SIG_FIELD_SETTING_EMAIL         = 7,
    SAPI_ENUM_SIG_FIELD_SETTING_SIGNED_SECTION = 8,
    SAPI_ENUM_SIG_FIELD_SETTING_FIELD_CONTAINER = 9,
    SAPI_ENUM_SIG_FIELD_SETTING_FIELD_MODE     = 10,
    SAPI_ENUM_SIG_FIELD_SETTING_CUSTOM1       = 30,
    SAPI_ENUM_SIG_FIELD_SETTING_CUSTOM2       = 31,
    SAPI_ENUM_SIG_FIELD_SETTING_CUSTOM3       = 32
} SAPI_ENUM_SIG_FIELD_SETTINGS_ID;
```

### **ENUM values**

#### *SAPI\_ENUM\_SIG\_FIELD\_SETTING\_NONE*

Not in use.

#### *SAPI\_ENUM\_SIG\_FIELD\_SETTING\_CLEAR\_PERMIT*

The conditions for clearing the signature field. Currently applicable only to legacy Word and PDF files.

#### *SAPI\_ENUM\_SIG\_FIELD\_SETTING\_TITLE*

The title of the person signing the field.

#### *SAPI\_ENUM\_SIG\_FIELD\_SETTING\_TITLE\_TAG*

The label of the title field.

#### *SAPI\_ENUM\_SIG\_FIELD\_SETTING\_SUGGESTED\_SIGNER*

The suggested signer of the signature field.

#### *SAPI\_ENUM\_SIG\_FIELD\_SETTING\_INSTRUCT\_TO\_SIGNER*

Instructions to the signer.

#### *SAPI\_ENUM\_SIG\_FIELD\_SETTING\_ALLOW\_REASON*

Whether to enforce entry of a reason.

#### *SAPI\_ENUM\_SIG\_FIELD\_SETTING\_EMAIL*

An email address of the signer.

#### *SAPI\_ENUM\_SIG\_FIELD\_SETTING\_SIGNED\_SECTION*

This parameter is relevant to InfoPath 2007/2010/2013 forms.

This enumerand identifies the signed section.

#### *SAPI\_ENUM\_SIG\_FIELD\_SETTING\_FIELD\_CONTAINER*

This parameter is relevant to InfoPath 2007/2010/2013 forms.

This enumerand identifies the signature field

*SAPI\_ENUM\_SIG\_FIELD\_SETTING\_FIELD\_MODE*

This parameter is relevant to InfoPath 2007/2010/2013 forms.  
This enumerand identifies the signature mode in InfoPath.

*SAPI\_ENUM\_SIG\_FIELD\_SETTING\_CUSTOM1*

Custom field 1.

*SAPI\_ENUM\_SIG\_FIELD\_SETTING\_CUSTOM2*

Custom field 2.

*SAPI\_ENUM\_SIG\_FIELD\_SETTING\_CUSTOM3*

Custom field 3.

## SAPI\_ENUM\_FIELD\_MODE

This enum type is relevant to InfoPath 2007/2010/2013 forms and represents the types of available signatures in an InfoPath form.

```
typedef SAPI_ENUM_FIELD_MODE {
    SAPI_ENUM_FIELD_MODE_NONE           = 0,
    SAPI_ENUM_FIELD_MODE_SINGLE         = 1,
    SAPI_ENUM_FIELD_MODE_COUNTER_SIGN   = 2,
    SAPI_ENUM_FIELD_MODE_CO_SIGN        = 3
} SAPI_ENUM_FIELD_MODE;
```

### ***ENUM values***

*SAPI\_ENUM\_FIELD\_MODE\_NONE*

Not in use.

*SAPI\_ENUM\_FIELD\_MODE\_SINGLE*

A single signature in the form.

*SAPI\_ENUM\_FIELD\_MODE\_COUNTER\_SIGN*

The multiple signatures in the form are dependent.

*SAPI\_ENUM\_FIELD\_MODE\_CO\_SIGN*

The multiple signatures in the form are independent.

## SAPI\_ENUM\_SIG\_EXT\_INFO\_FORMAT

The possible values of the extended information structure when using SAPI\_ENUM\_CONF\_ID\_RET\_EXT\_FIELD\_INFO as part of the configuration values. Please consult ARX before using this field.

```
typedef SAPI_ENUM_FIELD_EXT_INFO_FORMAT {  
    SAPI_ENUM_FIELD_EXT_INFO_FORMAT_NONE           = 0,  
    SAPI_ENUM_FIELD_EXT_INFO_FORMAT_OLD            = 1,  
    SAPI_ENUM_FIELD_EXT_INFO_FORMAT_V1            = 2  
} SAPI_ENUM_FIELD_EXT_INFO_FORMAT;
```

### ***ENUM values***

#### *SAPI\_ENUM\_FIELD\_EXT\_INFO\_FORMAT\_NONE*

Not in use.

#### *SAPI\_ENUM\_FIELD\_EXT\_INFO\_FORMAT\_OLD*

The record format is in a format corresponding to a CoSign pre 4.4 version.

#### *SAPI\_ENUM\_FIELD\_EXT\_INFO\_FORMAT\_V1*

The record format is a CoSign V4.4+ format. In this version, the information contains the custom fields.

## **SAPI\_ENUM\_TICKET\_CHECK\_STATUS**

Contact ARX support for more information.

## **SAPI\_ENUM\_ADVANCED\_SIG\_LEVEL**

The possible advanced signature conformance levels.

The levels are defined according to the XAdES or CAdES definitions.

To read more about XAdES, refer to <http://www.w3.org/TR/XAdES/>.

To read more about CAdES, refer to RFC 5126 documentation.

```
typedef SAPI_ENUM_ADVANCED_SIG_LEVEL {
    SAPI SAPI_ENUM_ADVANCED_SIG_LEVEL_NONE          = 0,
    SAPI_ENUM_ADVANCED_SIG_LEVEL_BES                = 1,
    SAPI_ENUM_ADVANCED_SIG_LEVEL_PES                = 2,
    SAPI_ENUM_ADVANCED_SIG_LEVEL_TS
} SAPI_ENUM_ADVANCED_SIG_LEVEL;
```

### *SAPI\_ENUM\_ADVANCED\_SIG\_LEVEL\_NONE*

No limitation on levels.

### *SAPI\_ENUM\_ADVANCED\_SIG\_LEVEL\_BES*

The basic advanced level of advanced signatures.

### *SAPI\_ENUM\_ADVANCED\_SIG\_LEVEL\_PES*

The basic policy-based advanced level of advanced signatures. This is not implemented yet in CoSign Signature Local.

### *SAPI\_ENUM\_ADVANCED\_SIG\_LEVEL\_TS*

The time-stamp-based advanced level of advanced signatures.

## **SAPI\_ENUM\_FILE\_HANDLE\_TYPE**

The possible types of file handles that can be used by SAPI.

```
typedef SAPI_FILE_HANDLE_TYPE {  
    SAPI_ENUM_FILE_HANDLE_NONE      = 0,  
    SAPI_ENUM_FILE_HANDLE_MEMORY    = 1,  
    SAPI_ENUM_FILE_HANDLE_FILE      = 2  
} SAPI_ENUM_FILE_HANDLE_TYPE;
```

*SAPI\_ENUM\_FILE\_HANDLE\_NONE*

Not in use.

*SAPI\_ENUM\_FILE\_HANDLE\_MEMORY*

Content of file is in memory.

*SAPI\_ENUM\_FILE\_HANDLE\_FILE*

Content of file is on the disk.

## **SAPI\_ENUM\_SIG\_CLEAR\_POLICY**

The policy to follow when trying to clear a signature field.

```
typedef enum SAPI_ENUM_SIG_CLEAR_POLICY_TYPE {  
    SAPI_ENUM_SIG_CLEAR_POLICY_ALWAYS      = 0,  
    SAPI_ENUM_SIG_CLEAR_POLICY_NEVER      = 1,  
    SAPI_ENUM_SIG_CLEAR_POLICY_SIGNER_BY_CN = 2  
} SAPI_ENUM_SIG_CLEAR_SIG_POLICY;
```

### *SAPI\_ENUM\_SIG\_CLEAR\_POLICY\_ALWAYS*

There are no restrictions when clearing a signature field.

### *SAPI\_ENUM\_SIG\_CLEAR\_POLICY\_NONE*

It is not possible to clear a signature field.

### *SAPI\_ENUM\_SIG\_CLEAR\_POLICY\_SIGNER\_BY\_CN*

Only the signer can clear the signature field.

## **SAPI\_ENUM\_AUTH\_MODE**

The types of authentication employed by the appliance.

```
typedef enum SAPI_ENUM_AUTH_MODE{  
    SAPI_ENUM_AUTH_MODE_NONE           = 0,  
    SAPI_AUTH_MODE_SSPI                 = 1,  
    SAPI_AUTH_MODE_VERIFY_USER_SRV_SIDE = 2,  
    SAPI_AUTH_MODE_SSPI_USRPWD         = 3,  
    SAPI_AUTH_MODE_VERIFY_DB_USER_SRV_SIDE = 4,  
    SAPI_AUTH_MODE_SAML_SRV_SIDE       = 5  
} SAPI_ENUM_AUTH_MODE;
```

### **SAPI\_ENUM\_AUTH\_MODE\_NONE**

Authentication is not used.

### **SAPI\_ENUM\_AUTH\_MODE\_SSPI**

Active Directory SSPI mode.

### **SAPI\_ENUM\_AUTH\_MODE\_VERIFY\_USER\_SRV\_SIDE**

User and password verified by server.

### **SAPI\_ENUM\_AUTH\_MODE\_SSPI\_USRPWD**

SSPI with user and password parameters.

### **SAPI\_ENUM\_AUTH\_MODE\_VERIFY\_DB\_USER\_SRV\_SIDE**

The user and his credentials are declared in the database.

### **SAPI\_ENUM\_AUTH\_MODE\_SAML\_SRV\_SIDE**

SAML ticket validation.

## **SAPI\_ENUM\_SERVER\_KIND**

The type of appliance.

```
typedef enum SAPI_ENUM_SERVER_KIND{  
    SAPI_ENUM_SERVER_KIND_NONE           = 0,  
    SAPI_ENUM_SERVER_KIND_STANDALONE     = 1,  
    SAPI_ENUM_SERVER_KIND_MASTER        = 2,  
    SAPI_ENUM_SERVER_KIND_SLAVE         = 3  
} SAPI_ENUM_AUTH_MODE;
```

### **SAPI\_ENUM\_AUTH\_SERVER\_KIND\_NONE**

No appliance is used.

### **SAPI\_ENUM\_AUTH\_SERVER\_KIND\_STANDALONE**

Reserved for future use.

### **SAPI\_ENUM\_AUTH\_SERVER\_KIND\_MASTER**

Primary appliance.

### **SAPI\_ENUM\_AUTH\_SERVER\_KIND\_SLAVE**

Alternate appliance.

## **SAPI\_ENUM\_DIRECTORY\_KIND**

The type of directory.

```
typedef enum SAPI_ENUM_DIRECTORY_KIND{  
    SAPI_NONE_DIRECTORY_KIND          = 0,  
    SAPI_ACTIVE_DIRECTORY_KIND        = 1,  
    SAPI_NDS_DIRECTORY_KIND           = 2,  
    SAPI_PUSH_DIRECTORY_KIND          = 3,  
    SAPI_LDAP_DIRECTORY_KIND          = 4  
} SAPI_ENUM_DIRECTORY_MODE;
```

### **SAPI\_NONE\_DIRECTORY\_KIND**

No directory is used.

### **SAPI\_ACTIVE\_DIRECTORY\_KIND**

Active Directory.

### **SAPI\_NDS\_DIRECTORY\_KIND**

This directory type is not supported.

### **SAPI\_PUSH\_DIRECTORY\_KIND**

Directory Independent mode.

### **SAPI\_LDAP\_DIRECTORY\_KIND**

LDAP directory.

## **SAPI\_ENUM\_PAGE\_ROTATE**

Rotation of a page inside a given document.

```
typedef enum SAPI_ENUM_PAGE_ROTATE{  
    SAPI_ROTATE_NONE           = 0,  
    SAPI_ROTATE_90             = 1,  
    SAPI_ROTATE_180            = 2,  
    SAPI_ROTATE_270            = 3  
} SAPI_ENUM_PAGE_ROTATE;
```

### **SAPI\_ROTATE\_NONE**

No rotation is employed.

### **SAPI\_ROTATE\_90**

Rotate 90 degrees clockwise.

### **SAPI\_ROTATE\_180**

Rotate 180 degrees.

### **SAPI\_ROTATE\_270**

Rotate 270 degrees clockwise.

## Appendix D: CoSign Signature Local – Signing/Verifying Constants and Flags

This appendix describes all the constants and flags defined in CoSign Signature Local.

### General Constants

Constant	Description	Value
EMPTY_FIELD_LABEL_MAX_LEN	The max length of the empty signature field label string.	128
SIGNATURE_FIELD_NAME_MAX_LEN	The max length of the signature field name.	128
DATE_FORMAT_MAX_LEN	The max length of the date format string.	32
TIME_FORMAT_MAX_LEN	The max length of the time format string.	32
GR_SIG_NAME_MAX_LEN	The max length of a graphical image name	124
NUMBER_OF_POSSIBLE_DRAWING_ELEMENTS	The maximum number of elements (for example, reason, or signer name) that can be displayed in a signature field.	7
GR_IMG_FLG_ALWAYS_ON_TOP	The graphical image selection dialog box will appear on top of other windows. Consult with ARX before setting this flag.	0x00000004
GR_IMG_FLG_EDIT_AUTO_OK	Automatic OK after capturing graphical image. Consult with ARX before setting this flag.	0x00000008
GR_IMG_FLG_HIDE_COLOR	Hide color selection button. Consult with ARX before setting this flag.	0x00000010
GR_IMG_FLG_FORCE_EDIT_DLG	Show the whole graphical image selection dialog box and not only the Capture Signature section. Consult with ARX before setting this flag.	0x00001000
COSIGN_BASIC_SLOT_STRING	The CSP provider.	"AR CoSign Appliance"
SAPI_MAX_SIG_INSTRUCT_LEN	The maximum number of alphanumeric characters in the instructions text.	256
PERFORM_IMAGE_REDUCTION_DEFAULT	Use the default image reduction behavior when uploading a graphical image to DocuSign Signature Appliance.	0xFFFFFFFF
TOKEN_ID_MAX_LENGTH	The length of the token used to identify the appliance used in this SAPI session.	50
NOT_PERFORM_IMAGE_REDUCTION	Do not perform image reduction when uploading a graphical image to DocuSign Signature Appliance.	0

Constant	Description	Value
AR_SF_LOCATOR_MAX_STRING_LENGTH	This flag is relevant when specifying signature field location in PDF files. The maximum length of the field locator string.	256
AR_LOCATOR_TEXT_DIRECTION_AS_PAGE_ROTATION	The orientation of each field locator string is the same as the orientation of the page on which it is located. This parameter should be used only if at least one of the pages is rotated.	0x00000001
SERVER_SERIAL_NUMBER_LEN	The maximum number of characters in the appliance's serial number.	16

## Flags when Enumerating Signature Fields

The following values are used as flags when enumerating signature fields.

Flag	Description	Value
AR_INCLUDE_LEGACY_DOC_FIELDS_FLAG	Include also AR legacy fields in the signature field enumeration.	0x00000001
AR_SAPI_PDF_USE_SIGNER_GMT_OFFSET	Retrieve signature time according to the signer's GMT offset.	0x00000002
AR_SAPI_PDF_ALLOW_FIELD_LOCATOR	Enumerate locators as part of the field enumeration process.	0x00000004

## Flags used by the SAPILogonEx Function

The following values are used as flags by the [SAPILogonEx](#) Function.

Flag	Description	Value
SAPI_LOGON_FLAG_TYPE_WRITE_OP	Enable the open session to update user account information, such as the user's graphical image information. This means that the open session will be based on communication with the primary appliance.	0x00000001
SAPI_LOGON_FLAG_AUTH_MODE_SSPI	Instruct SAPI to perform an Active Directory SSPI login	0x00000100
SAPI_LOGON_FLAG_AUTH_MODE_VERIFY_USER_SRV_SIDE	Instruct SAPI to perform a server side user/pwd login when Active Directory or LDAP is used.	0x00000200
SAPI_LOGON_FLAG_AUTH_MODE_SSPI_USRPWD	Instruct SAPI to perform an Active Directory SSPI client side login.	0x00000400
SAPI_LOGON_FLAG_AUTH_MODE_VERIFY_DB_USER_SRV_SIDE	Instruct SAPI to perform an Active Directory server side login.	0x00000800
SAPI_LOGON_FLAG_AUTH_MODE_SAML_SRV_SIDE	Instruct SAPI to perform SAML based authentication.	0x00001000
SAPI_LOGON_FLAG_TYPE_DIRECT	Instruct SAPI to connect directly to the Appliance. This type of connection achieves better performance rates.	0x10000000
SAPI_LOGON_FLAG_EXTERNAL_COSIGN_SRV	Instruct SAPI to connect directly to an external Appliance, identified by the Token ID set in the <a href="#">SAPISetTokenID</a> function. Note that a direct connection, whether to an internal or external appliance, achieves better performance rates than an indirect connection.	0x20000000

## Flags used by the SAPIGetTokenInfo Function

The following values are used as flags by the [SAPIGetTokenInfo](#) function.

Flag	Description	Value
SAPI_TOKEN_FLAG_TYPE_WRITE_OP	The appliance is able to update information (meaning it is a primary appliance).	0x00000001

## Flags when Enumerating Certificates

The following values are used as flags when enumerating certificates.

Flag	Description	Value
PERFORM_EXTENDED_VALIDATION	To be used as part of the certificate enumeration function for enabling certificate path validation. Certificates that do not pass the validation process are not included in the certificate enumeration. This flag cannot be used when displaying a certificate list. In this case, you should use the SAPI_ENUM_CONF_ID_EXTENDED_VALIDATION configuration value.	0x00000001

## Flags when Displaying Certificate List

The following values are used as flags when displaying a certificate list.

Flag	Description	Value
DISPLAY_ALL_CERTS_IN_DIALOG	Display all available certificates (even if there is only one certificate).	0
DONT_DISPLAY_DIALOG_FOR_SINGLE_CERT	If there is only one available certificate, the function does not display the dialog, and returns the certificate's handle.	1

## Flags when Enumerating or Using Graphical Signatures

The following values are used as flags when enumerating graphical signatures.

Flag	Description	Value
AR_GR_FLAG_ALL_TYPE	Use this flag to retrieve all types of graphical images.	0x00FF0000
AR_GR_FLAG_IMAGE_TYPE	Use this flag to retrieve graphical signatures. This flag can also be used when creating a new graphical image.	0x00010000
AR_GR_FLAG_INITIALS_TYPE	Use this flag to retrieve initials images. This flag can also be used when creating a new graphical image.	0x00020000
AR_GR_FLAG_LOGO_TYPE	Use this flag to retrieve logo images. This flag can also be used when creating a new graphical image.	0x00040000
AR_GR_FLAG_VOLATILE_TYPE	This type identifies a volatile graphical signature (which is not actually kept in the user account in the appliance).	0x00100000

## Flags when Using the Signing Ceremony API

The following values are used as flags when using the signing ceremony API.

Value	Meaning	Value
RESERVED_SC_ENABLE_THREAD	Use this flag to execute the signing ceremony in a thread. Consult ARX before using this option.	0x00000001
RESERVED_SC_STORAGE_BASIC_CONFIG	Use this flag to specify that the user is restricted to using the system-wide configuration. For example, the reasons list cannot be changed by the user.	0x00000002
AR_DISPLAY_ALLOW_TIMESTAMP	Used by the application to notify the signing ceremony that it allows the display of a time stamp checkbox	0x02000000

## Flags when Creating a Signature Field in PDF Files

The following values are used as flags when creating a new signature field in PDF documents.

These flags can be passed as part of the *Flags* input parameter of the [SAPISignatureFieldCreateEx](#) and

SAPISignatureFieldCreateSignEx\_functions.

Flag	Description	Value
AR_PDF_FLAG_FIELD_NAME_SET	Enables manually setting the name of the signature field.	0x00000080

## Flags when Signing PDF Files

The following values are used as flags when signing Adobe documents.

These flags can be passed as part of the *Flags* input parameter of the

SAPISignatureFieldCreateSignEx and the [SAPISignatureFieldSignEx](#) functions.

Flag	Description	Value
AR_PDF_FLAG_FILESIGN_LINE	Use the vectorial representation rather than the BMP representation when writing the graphical signature image to the file.	0x00000010
AR_PDF_FLAG_FILESIGN_VERTICAL	Split the graphical signature image and the other text elements vertically.	0x00000020
AR_PDF_FLAG_CERTIFY	The signature operation will use the PDF's certify operation instead of a regular digital signature operation. The certify restriction level will be level 2.	0x00000040
AR_PDF_FLAG_CERTIFY_2	The signature operation will use the PDF's certify operation instead of a regular digital signature operation. In the PDF certify operation there are three levels of restrictions that are implemented once the document is signed. This flag should be used in combination with the following to achieve all three levels of restrictions: <ul style="list-style-type: none"> <li><input type="checkbox"/> Restriction Level 1 – AR_PDF_FLAG_CERTIFY_1</li> <li><input type="checkbox"/> Restriction Level 2 - AR_PDF_FLAG_CERTIFY_2</li> <li><input type="checkbox"/> Restriction Level 3 - AR_PDF_FLAG_CERTIFY_1   AR_PDF_FLAG_CERTIFY_2</li> </ul>	0x00000040
AR_PDF_FLAG_CERTIFY_1	See the description of AR_FLAG_CERTIFY_2	0x40000000
AR_PDF_FLAG_FILESIGN_FORCE_GRIMG_ERR	If a graphical image is required and the relevant user does not have any graphical image, determines whether to stop the signature operation with an error.	0x80000000

## Flags when Creating a Signature Field in TIFF Files

The following values are used as flags when creating a new signature field in TIFF documents.

These flags can be set as part of the signature field's *Flags* attribute in the [SAPI\\_SIG\\_FIELD\\_SETTINGS](#) structure.

Flag	Description	Value
AR_TIFF_FLAG_FIELD_DRAW_BANNER_METHOD	Indicates the field drawing method is set to banner style.	0x00000100
AR_TIFF_FLAG_FIELD_DONOT_CLEAR_EMBEDDED_METHOD	Indicates that the visible field cannot be cleared. This directs CoSign Signature Local to not keep the image that is covered by the newly generated visible signature.	0x00000200

## Flags when Signing TIFF Files

The following values are used as flags when signing a TIFF document.

These flags can be passed as part of the *Flags* input parameter of the [SAPISignatureFieldSignEx](#) function.

Flag	Description	Value
AR_TIFF_FLAG_SIGNING_FLAGS_MASK	Reserved – not in use.	0x0000003F
AR_TIFF_FLAG_PACKBITS_COMPRESSION	Use pack bit compression for the first display page (banner). The actual compression mode is PACKBITS_COMPRESSION.	0x00000002
AR_TIFF_FLAG_GROUP4FAX_COMPRESSION	Use pack bit compression for the first display page (banner). The actual compression mode is GROUP4FAX.	0x00000004

## Flags when Using a Signature Field in PDF Documents

The following values are used as flags when using signature field in PDF documents.

These flags can be set as part of the signature field's *Flags* attribute in the [SAPI\\_SIG\\_FIELD\\_SETTINGS](#) structure.

Flag	Description	Value
AR_PDF_DISABLE_REASON	Reason is disabled	0x00000001
AR_PDF_SIGN_HORIZONTAL	Signature field's visible information is displayed horizontally.	0x00000002
AR_PDF_SIGN_VERTICAL	Signature field's visible information is displayed vertically.	0x00000004
AR_PDF_FIELD_LOCATOR	The field is based on a locator.	0x00000008
AR_PDF_FLAG_CENTER_ALIGN_TEXT_LINES	Whether to align the text in the visible signature to the center. This functionality is relevant only if the new PDF processing method is used.	0x00400000
AR_PDF_FLAG_NO_IMAGE_RESIZE	Whether to keep the size of the graphical signature as-is inside the signature. This functionality is relevant only if the new PDF processing method is used.	0x00800000

## Flags when Using a Signature Field in Word Documents

The following values are used as flags when using signature field in Word documents.

These flags can be set as part of the signature field's *Flags* attribute in the [SAPI\\_SIG\\_FIELD\\_SETTINGS](#) structure.

Flag	Description	Value
AR_WORD_DUMMY_FIELD_CREATION	Reserved – Not in use.	0x00000001
AR_WORD_XP_COMPATIBLE_FLAG	Add a word XP compatible signature when signing this field. This means that the signature can be verified by Word XP without the AR word plug-in.	0x00000001
AR_WORD_OUTLOOK_INCOMPATIBLE	Include the file's summary information in the data to be signed. Note that if this flag is set and the file is sent via Outlook, Outlook changes the file's summary information and thus invalidates the signature.	0x00000002
AR_WORD_SHAREPOINT_2007_COMPATIBLE	Include the SharePoint related information in the data to be signed. Note that if this flag is set and the file is used by SharePoint, SharePoint changes some of the file's summary information and thus invalidates the signature.	0x00000004

## Flags when Creating a Signature Field in Office 2007/2010/2013 Documents

The following values are used as flags when creating a new signature field in Office 2007/2010/2013 documents.

These flags can be passed as part of the *Flags* input parameter of the [SAPISignatureFieldCreateEx](#) and

SAPISignatureFieldCreateSignEx functions.

Flag	Description	Value
AR_MS_SIGNATURE_LINE_ONLY	If this flag is set, the created empty signature field is a Microsoft field and not an ARX Signature Line Provider field.	0x00000001
AR_SAPI_SIG_CREATE_WITH_EMFONLY_FLAG	Used to avoid a mismatch in signature appearance in Office vs. SAPI.	0x00200000

## Flags when Signing Word Documents

The following values are used as flags when signing Word documents.

These flags can be passed as part of the *Flags* input parameter of the [SAPISignatureFieldSignEx](#) function.

Flag	Description	Value
AR_WORD_REDRAW_SIG_METAFILE_FLAG	In case of a single signature in a file, the signature appearance is written to the file, so that the signature can be seen even in a version of Word without the AR plug-in. It cannot, however, be verified. Note that by default this feature is disabled. Contact ARX support for instructions on enabling this feature.	0x00000001
AR_WORD_OPERTAION_FROM_ADDIN_FLAG	Reserved – not in use.	0x00000002
AR_WORD_CLEAR_DEPENDENT_SIGNATURES_FLAG	Clear all the signatures in the signature fields that depend on this field prior to signing. Note that this flag can also be used for <a href="#">SAPISignatureFieldClear</a> , <a href="#">SAPISignatureFieldRemove</a> , and <a href="#">SAPISignatureFieldCreate</a> .	0x00000004
AR_WORD_CLEAR_ALL_SIGNATURES_FLAG	Clear the signatures in all the signature fields in the document prior to signing. Note that this flag can also be used for <a href="#">SAPISignatureFieldClear</a> , <a href="#">SAPISignatureFieldRemove</a> , and <a href="#">SAPISignatureFieldCreate</a> .	0x00000008

## **Flags when Signing XML Documents**

The following values are used as flags when signing XML documents.

These flags can be passed as part of the *Flags* input parameter of the

SAPISignatureFieldCreateSignEx function.

Flag	Description	Value
AR_XML_XML_SIG_TYPE_ENVELOPED	<p>Use Enveloped XML signature format.</p> <p><b>Note:</b> You can specify either Enveloped XML signature format, or Enveloping XML signature format, not both.</p> <p>The Enveloped XML signature format is the default signature format when no flag is specified.</p>	0x00000001
AR_XML_XML_SIG_TYPE_ENVELOPING	<p>Use Enveloping XML signature format.</p> <p><b>Note:</b> You can specify either Enveloped XML signature format, or Enveloping XML signature format, not both.</p>	0x00000002
AR_XML_SIG_TYPE_XMLDIGSIG	<p>Use a regular XML digital signature.</p> <p><b>Note:</b> You can specify either a regular XML digital signature, or an advanced-BES XML signature, not both.</p>	0x00000008
AR_XML_SIG_TYPE_XADES_BES	<p>Use an advanced-BES XML digital signature.</p> <p><b>Note:</b> You can specify either a regular XML digital signature, or an advanced-BES XML signature, not both.</p> <p>The advanced-BES XML signature is the default XML signature when no flag is specified.</p>	0x00000010

## Time-Stamping/OCSP and Miscellaneous Signature-Related Flags

The following values are used when involving a time stamping service or OCSP as part of the digital signature operation. The table also includes various miscellaneous constants.

Flag	Description	Value
AR_GR_SIG_ENABLE_STS, AR_SAPI_SIG_ENABLE_STS	Enable the time stamping service.	0x00000100
AR_GR_SIG_DISABLE_STS, AR_SAPI_SIG_DISABLE_STS	Disable the time stamping service.	0x00000200
AR_SAPI_SIG_HASH_ONLY	The value for buffer signing is the hash itself and not the data.	0x00000400
AR_SAPI_SIG_PDF_REVOCATION	A flag for the signing operation, indicating to embed revocation information when signing PDF-style. Currently supported only in PDF files.	0x00001000
AR_SAPI_SIG_CADES_REVOCATION	A flag for the signing operation, indicating to embed revocation information when signing. Currently NOT supported.	0x00002000
AR_SAPI_SHA256_FLAG	A flag for the signing operation, indicating to use SHA256 for signing. Note that verification can fail if the verifying party does not have the SHA256 capability.	0x00004000
AR_SAPI_SHA384_FLAG	A flag for the signing operation, indicating to use SHA384 for signing. Note that verification can fail if the verifying party does not have the SHA384 capability.	0x00008000
AR_SAPI_SHA512_FLAG	A flag for the signing operation, indicating to use SHA512 for signing. Note that verification can fail if the verifying party does not have the SHA512 capability.	0x00010000
AR_SAPI_SIG_ENFORCE_NO_LOGOUT	When this flag is used as part of a digital signature operation, the signing user will remain logged-in after completing the signing operation. This flag is not relevant for cases where the session is dynamic (based on using SAPILogon and SAPILogoff calls).	0x00020000
AR_SAPI_SIG_ENFORCE_LOGOUT	When this flag is used as part of a digital signature operation, the signing user will be automatically logged off. This flag is not relevant for cases where the session is dynamic (based on using SAPILogon and SAPILogoff calls).	0x00040000

Flag	Description	Value
AR_SAPI_SIG_PKCS1	When this flag is used as part of a digital signature operation, the generated signature is based on PKCS#1 format. This option is relevant only to buffer signing.	0x00080000
AR_SAPI_ENFORCE_SHA1_FLAG	Use this flag to enforce a generation of a SHA1 signature.	0x00100000
PERFORM_EXTENDED_VALIDATION	When this flag is used, certificates that do not pass the validation test cannot be used for signature operation. This flag can be used as part of the certificate enumeration function for enabling certificate path validation.	0x00000001

## Flags when Verifying a Signature

The following values are used as flags when verifying a signature of a signature field of any file type ([SAPISignatureFieldVerify](#)), or a signature of raw data ([SAPIBufferVerifySignature](#), [SAPIBufferVerifySignatureInit](#)).

Flag	Description	Value
DONT_USE_FIELD_TIME_FOR_CERT_VERIFY	If the signature time does not exist in the PKCS#7 signature blob, use the current time for validating the certificate, rather than the signature field signing time or the time that is passed as an input parameter.	0x00000001
AR_SAPI_SIG_HASH_ONLY	The value for buffer signature validation is the hash itself and not the data.	0x00000400
AR_SAPI_SHA256_FLAG	A flag for the signing operation, indicating to use SHA256 for signing. Relevant only if the hash is given.	0x00004000
AR_SAPI_SHA384_FLAG	A flag for the signing operation, indicating to use SHA384 for validation. Relevant only if the hash is given.	0x00008000
AR_SAPI_SHA512_FLAG	A flag for the signing operation, indicating to use SHA512 for validation. Relevant only if the hash is given.	0x00010000
AR_SAPI_ENFORCE_SHA1_FLAG	A flag for the signing operation, indicating to use SHA1 for validation. Relevant only if the hash is given.	0x00100000
AR_SAPI_SIG_PKCS1	When this flag is used as part of a digital signature verification operation, the given signature is based on PKCS#1 format. This option is relevant only to buffer signature verification.	0x00080000

## Flags when Clearing or Removing Signature Fields

The following values are used as flags when clearing or removing signature fields.

Flag	Description	Value
AR_WORD_CLEAR_DEPENDENT_SIGNATURES_FLAG	Clear all the signatures in the signature fields that depend on this field, prior to clearing or removing this field.	0x00000004
AR_WORD_CLEAR_ALL_SIGNATURES_FLAG	Clear the signatures in all the signature fields in the document prior to clearing or removing this field.	0x00000008

## Graphical Signature Image Related Flags

The following flags are used when initializing the graphical image enumeration operation, or to filter the images that are shown in the GUI dialog box when calling the [SAPIGraphicSigImageGUISelect](#) or [SAPISigningCeremonyGUI](#) functions.

Flag	Description	Value
AR_GR_SIG_DATA_FORMAT_MONO_BMP	Monochrome BMP image.	0x00000001
AR_GR_SIG_DATA_FORMAT_COLOR_BMP	Colored BMP image.	0x00000002
AR_GR_SIG_DATA_FORMAT_COLOR_1_BMP	Colored BMP image, but only 2 colors are in use.	0x00000004
AR_GR_SIG_DATA_FORMAT_JPG	JPEG image.	0x00000008
AR_GR_SIG_DATA_FORMAT_TIFF	TIFF image.	0x00000010
AR_GR_SIG_DATA_FORMAT_GIF	GIF format.	0x00000020
AR_GR_SIG_DATA_FORMAT_UNKNOWN	Unknown format.	0x00000000
AR_GR_SIG_DATA_FORMAT_ALL	Get all formats.	0xFFFFFFFF
CAP_DEV_DLG_ID_AUTOMATIC	Capturing device – automatic selection.	0
CAP_DEV_DLG_ID_GUI_OR_AUTOMATIC	Capturing device – manual selection.	1
CAP_DEV_DLG_ID_VIRTUAL	Capturing device – external DLL.	2
CAP_DEV_DLG_ID_TOPAZ	Capturing device – Topaz pad.	3
CAP_DEV_DLG_ID_EPAD	Capturing device – Interlink pad.	4
CAP_DEV_DLG_ID_TABLET_PC	Capturing device – Tablet PC or Mouse.	5
CAP_DEV_DLG_ID_TEXT	Capturing device – Script font Mouse.	6

**Note:** CoSign Signature Local supports only BMP and JPG formats in its signature field signing function.

The following is the value of the flags parameter in [SAPIGraphicSigImageInfoGet](#).

Flag	Description	Value
AR_GR_SIG_GET_INTERNAL_ALLOC	Let CoSign Signature Local allocate the memory for the graphical signature image.	0x00000001

## Custom Fields-Related Constants

Constant	Description	Value
MAX_NUM_OF_CUSTOM_FIELDS	The maximum number of customer fields in a signature field.	15
CURRENT_CUSTOM_FIELDS_STRUCT_VERSION	The current structure version of the custom field.	1

## Appendix E: CoSign Signature Local – Signing/Verifying Error Messages

This appendix describes the error codes returned by CoSign Signature Local.

Error Code	Description	Numeric Value
SAPI_OK	Function returned with no error.	0
SAPI_GENERAL_ERROR	A general failure message returned from CoSign Signature Local.	0x90010001
SAPI_ERR_FAILED_TO_INIT	Failed to initialize the SAPI library. Can be caused by installing an old CoSign Signature Local version on a machine with a newer client.	0x90020100
SAPI_ERR_FAILED_TO_INIT_REG_VALUE	Cannot read the CoSign Signature Local installation path. The Install_path value cannot be read from the registry.	0x90020101
SAPI_ERR_FAILED_TO_INIT_REG_KEY	Cannot read the CoSign Signature Local installation path. The Install_path key cannot be found.	0x90020102
SAPI_ERR_FAILED_TO_INIT_LOAD_MODULE	Failed to initialize the SAPI library. Failed to load the SAPI DLL.	0x90020103
SAPI_ERR_FAILED_TO_FIND_SERVER	Cannot find a server.	0x90020110
SAPI_ERR_FAILED_TO_CONNECT_TO_SERVER	Failed to connect to the server.	0x90020111
SAPI_ERR_SERVER_COMM_ERROR	Request from DocuSign Signature Appliance failed due to a communication error.	0x90020112
SAPI_ERR_SERVER_BLOCKED	Request from DocuSign Signature Appliance failed due to due client timeout.	0x90020113

Error Code	Description	Numeric Value
SAPI_ERR_USER_IS_NOT_LOGON	User is trying to perform an operation that requires authentication before calling the logon function.	0x90020120
SAPI_ERR_FAILED_TO_LOGON	Failed to logon to DocuSign Signature Appliance. Verify that the user name and password are correct.	0x90020130
SAPI_ERR_HANDLE_ALREADY_LOGGED_ON	SAPI session is active for another user. This error may also be returned when a second call to logon is attempted without a logoff. The error may also occur when a call to logon is made after the SAPI session was used to select objects under the credentials of the default user.	0x90020131
SAPI_ERR_LL_FAILURE_IN_LOGON	Failed to allocated resources for a new SAPI-LOGIN session (dynamic slot).	0x90020133
SAPI_ERR_ALREADY_LOCKED	The user session is already locked for third-party applications.	0x90020137
SAPI_ERR_LOCK_NOT_REQUIRED	No need for lock for third-party applications.	0x90020138
SAPI_ERR_FAILED_TO_UPDATE_CREDENTIAL	An attempt to change the user credentials failed.	0x90020150
SAPI_ERR_FAILED_TO_DELETE_USER	CoSign Signature Local failed to delete a user from DocuSign Signature Appliance. Call <a href="#">SAPIUMExtendedLastErrorGet</a> to retrieve the specific error code returned from DocuSign Signature Appliance.	0x90020160

Error Code	Description	Numeric Value
SAPI_ERR_FAILED_TO_BEGIN_SYNC	Failed to initialize a synchronization process. Call <a href="#">SAPIUMExtendedLastErrorGet</a> to retrieve the specific error code returned from DocuSign Signature Appliance.	0x90020170
SAPI_ERR_FAILED_TO_END_SYNC	Failed to end a synchronization process. Ending a synchronization process directs DocuSign Signature Appliance to delete users that are not in sync. When this function returns an error, the synchronization process is cancelled. Call <a href="#">SAPIUMExtendedLastErrorGet</a> to retrieve the specific error code returned from DocuSign Signature Appliance.	0x90020180
SAPI_ERR_FAILED_ACTIVATE_USER	Failure in user activation process.	0x900201A0
SAPI_ERR_CRED_AND_NEW_CRED_CANNOT_BE_EQUAL	A new password identical to the old password was entered when changing a password or activating a user account.	0x900201A1
SAPI_ERR_USER_ALREADY_ACTIVATED	The user is already activated.	0x900201A2
SAPI_ERR_SESSION_GET_FAILED	A technical problem occurred.	0x900201A3
SAPI_ERR_FAILED_TO_GET_CA_INFO	Failed to get CA information. DocuSign Signature Appliance returns this error if the installed CA does not support the requested information type.	0x90020190
SAPI_ERR_FAILED_TO_LOGOFF	Failed to logoff from a SAPI session.	0x900201b0

Error Code	Description	Numeric Value
SAPI_ERR_FAILED_TO_WRITE_USER	Failed to add or sync a user to DocuSign Signature Appliance. Call <a href="#">SAPIUMExtendedLastErrorGet</a> to retrieve the specific error code returned from DocuSign Signature Appliance.	0x900201d0
SAPI_ERR_FAILED_TO_WRITE_USER_EXIST	Failed to add a user to DocuSign Signature Appliance. A user with the same ID already exists.	0x900201d2
SAPI_ERR_FAILED_TO_WRITE_USER_EXCEED_LICENSE	Failed to add a user to DocuSign Signature Appliance because the maximum number of users provided under the license has been reached. Either delete users that are not active, or contact ARX to get a license that allows for more users.	0x900201d4
SAPI_ERR_FAILED_TO_VERIFY_USRPWD	Failed to verify the user password. Passwords should be in wide character representation. Password length in bytes includes the null terminator (two bytes in wide char representation).	0x900201e0
SAPI_ERR_KEY_LOCKED	The user key is locked and cannot be used for signing.	0x900201e1
SAPI_ERR_USR_LOCKED	The user is locked and cannot logon.	0x900201e2
SAPI_ERR_USR_PASSWORD_EXPIRED	The user's password has expired.	0x900201e3
SAPI_ERR_USR_DISABLED	The user is in a disabled state.	0x900201e4
SAPI_ERR_USR_NOT_ACTIVATED	The user was not activated yet.	0x900201E5
SAPI_ERR_USER_PWD_DO_NOT_FIT_POLICY	The user's password does not fit the configured password policy.	0x900201E6

<b>Error Code</b>	<b>Description</b>	<b>Numeric Value</b>
SAPI_ERR_USER_PWD_IN_MIN_VALIDITY_PERIOD	The minimum time for a password change has not passed yet.	0x900201E7
SAPI_ERR_USR_DISABLED	The user is in a disabled state	0x900201e4
SAPI_ERR_FAILED_TO_UPDATE_USER	Failed to update the user record.	0x90020201
SAPI_ERR_FAILED_TO_RESET_COUNTER	Failed to reset the user's counter.	0x90020210
SAPI_ERR_FAILED_TO_GET_USER	Failed to retrieve user's information.	0x90020215
SAPI_ERR_SESSION_IS_NULL	The session parameter cannot be NULL.	0x90020220
SAPI_ERR_INVALID_SES_HANDLE	The session handle used is either not initialized or is corrupt.	0x90020222
SAPI_ERR_NO_SUCH_USER	No such user exists in DocuSign Signature Appliance.	0x90020300
SAPI_ERR_TOKEN_NOT_SUPPORTED	DocuSign Signature Appliance is the only supported token for this operation.	0x90030100
SAPI_ERROR_NO_MORE_ITEMS	There are no more graphical images or certificates to retrieve. This return code is usually used to indicate the end of an objects list and should not be interpreted as indicating an error.	0x90030103
SAPI_CANT_DELETE_ZOMBIE_USERS	Failed to delete zombie users. Zombie users are users that were not touched while synchronizing DocuSign Signature Appliance with the user directory. Zombie users should be deleted when the sync operation ends.	0x90030110

Error Code	Description	Numeric Value
SAPI_ERR_FAILED_TO_ENUM_USERS	The users enumeration operation failed.	0x90030120
SAPI_FAILED_TO_ALLOCATE_MEMORY	A CoSign Signature Local operation required dynamic memory allocation and the allocation failed. Verify that there are no memory leaks or other resource problems.	0x90030130
SAPI_COM_FAILED_TO_ALLOCATE_MEMORY	A SAPICOM operation required dynamic memory allocation and the allocation failed. Verify that there are no memory leaks or other resource problems.	0x90030131
SAPI_ERR_FAILED_TO_GET_USER_INFO	Failed to get a user information structure. Call <a href="#">SAPIUMExtendedLastErrorGet</a> to retrieve the error code that was returned from DocuSign Signature Appliance.	0x90030140
SAPI_FUNCTION_NOT_SUPPORTED	This function is not supported, although defined by the CoSign Signature Local header.	0x90030150
SAPI_COM_FUNCTION_NOT_SUPPORTED	This function is not supported, although defined by the CoSign Signature Local header.	0x90030151
SAPI_NO_MORE_USERS	The user enumeration operation finished and there are no more users to retrieve. This return code is usually not an error but an indication that all the users were already retrieved.	0x90030160

Error Code	Description	Numeric Value
SAPI_ERR_FAILED_TO_CANCEL_SYNC	The cancellation of the synchronization operation of the DocuSign Signature Appliance database with the external user directory failed. Do not call <a href="#">SAPIUMUsersSyncEnd</a> before initializing a new synchronization operation.	0x90030170
SAPI_ILLEGAL_CA_INFO_TYPE	The CA information type that was required is not one of the supported values.	0x90030180
SAPI_ERR_FAILED_TO_FINALIZE	Failed to release the SAPI library.	0x90030190
SAPI_ERR_CONTEXT_NOT_INITIALIZED	The context that is used is either not initialized by the relevant Init function or is corrupt.	0x900301a0
SAPI_ERR_TOKEN_HANDLE_NOT_INITIALIZED	The handle that is used is either not initialized by the relevant function or is corrupt.	0x900301b0
SAPI_ERR_CONF_VALUE_NOT_FOUND	The configuration value requested by the function is not one of the supported values.	0x900301c0
SAPI_ERR_FAILED_TO_GET_CONF_VALUE	Failed to get a configuration value. Make sure the length is correct.	0x900301c1
SAPI_ERR_FAILED_TO_SET_CONF_VALUE	Failed to set a configuration value. Make sure the type is correct.	0x900301c2
SAPI_ERR_NULL_VALUE_IS_NOT_ALLOWED	A NULL value is not allowed for one or more of the parameters.	0x900301d0

<b>Error Code</b>	<b>Description</b>	<b>Numeric Value</b>
SAPI_ERR_FAILED_TO_SET_CERT	Failed to set a certificate as a default certificate for further signing operations.	0x900301e0
SAPI_ERR_FAILED_TO_GET_CERT	Failed to get the default certificate. Get the extended return code for more information about the source of the problem.	0x900301e1
SAPI_ERR_CERT_HANDLE_NOT_INITIALIZED	The certificate handle used is either not initialized or is corrupt.	0x900301f0
SAPI_ERR_PARAMETER_LEN_IS_TOO_SHORT	The length allocated for the output parameter is too short. Usually the required length is returned in the corresponding length parameter.	0x90030210
SAPI_COM_ERR_PARAMETER_LEN_IS_TOO_SHORT	The length that was allocated for the output parameter is too short. Usually the required length is returned in the corresponding length parameter.	0x90030211
SAPI_ERR_UNSUPPORTED_CERT_FIELD	The value of the certificate field that is required is not one of the supported values.	0x90030220
SAPI_ERR_FAILED_TO_GET_CERT_FIELD	Failed to retrieve the certificate field.	0x90030230
SAPI_ERR_FAILED_TO_FIND_AUTHORITY_KEY_ID	Failed to retrieve the authority key ID.	0x90030231
SAPI_ERR_CERT_INVALID_OR_EMPTY_CHAIN	Failed to retrieve a certificate chain.	0x90030232
SAPI_ERR_FAILED_TO_FIND_OCSP_URL	Failed to get the URL of the OCSP server.	0x90030233
SAPI_ERR_CERT_REVOKED_BY_OCSP	The certificate is revoked using the OCSP service.	0x90030234

Error Code	Description	Numeric Value
SAPI_ERR_OCSP_CERT_GENERAL_ERROR	The OCSP check, performed as part of the signature operation, is reporting a general error.	0x90030235
SAPI_ERR_PARAMETER_LEN_IS_TOO_LONG	Parameter length is too long.	0x90030236
SAPI_ERR_OCSP_CERT_FAILED_TO_GET	Failed to get the OCSP response for the relevant user certificate.	0x90030237
SAPI_ERR_FAILED_TO_GET_CERT_FIELD	Failed to retrieve the certificate field.	0x90030230
SAPI_ERR_FAILED_TO_INIT_CERT_ENUM	Failed to initialize the certificate enumeration operation.	0x90030240
SAPI_ERR_FAILED_TO_CONT_CERT_ENUM	Failed in enumerating certificates.	0x90030250
SAPI_ERR_FAILED_TO_SIGN_BUF	Failed to sign buffer. This usually occurs when there is no access to the private key. For example, when prompt for sign is set and an incorrect password is entered, or the process is unattended and the password cannot be entered.	0x90030270
SAPI_ERR_FAILED_TO_VERIFY_BUF	Failed to verify buffer.	0x90030271
SAPI_ERR_FAILED_TO_SIGN_XP_STYLE	Failed to sign a word file with an office compatible signature.	0x90030273
SAPI_ERR_FAILED_TO_INIT_SIGN_BUF	Failed to initialize the continuous sign buffer operation.	0x90030280
SAPI_ERR_FAILED_TO_INIT_VERIFY_BUF	Failed to initialize the continuous verify buffer operation.	0x90030281
SAPI_ERR_FAILED_TO_CONT_SIGN_BUF	Failed in maintaining the continuous sign buffer operation.	0x90030290

Error Code	Description	Numeric Value
SAPI_ERR_FAILED_TO_CONT_VERIFY_BUF	Failed in maintaining the continuous verify buffer operation.	0x90030291
SAPI_ERR_FAILED_TO_END_SIGN_BUF	Failed to sign buffer. This usually occurs when there is no access to the private key. For example, when prompt for sign is set and an incorrect password is entered, or the process is unattended and the password cannot be entered.	0x900302a0
SAPI_ERR_FAILED_TO_END_VERIFY_BUF	Failed to verify buffer.	0x900302a1
SAPI_ERR_TOO_MANY_CERTS_TO_SELECT_FROM	Failed to get the default certificate. The user has more than one certificate and CoSign Signature Local cannot determine which one should be used as the default.	0x900302b0
SAPI_ERR_CERT_FIELD_VALUE_NOT_FOUND	Failed to find the required field in the certificate.	0x900302c0
SAPI_ERR_NO_CERT_TO_SELECT_FROM	Failed to get the default certificate. The user has no valid certificate in the store.	0x900302d0
SAPI_ERR_FAILED_TO_BUILD_CERT_CHAIN	Failed to build a certificate chain.	0x900302e0
SAPI_ERR_GUI_SELECT_CERT_RETURNED_NO_CERT	The Select Certificate GUI dialog box found no certificate.	0x900302f0
SAPI_ERR_GUI_SELECT_SELECTED_CERT_NOT_FOUND	Unable to retrieve the certificate that was selected by the user via the Select Certificate dialog box.	0x900302f1
SAPI_ERR_GUI_SELECT_CERT_OPERATION_CANCELLED	The user pressed the <b>Cancel</b> button in the Select Certificate dialog box, and no certificate was selected.	0x900302f2

Error Code	Description	Numeric Value
SAPI_ERR_UNSUPPORTED_GRAPHICAL_IMAGE_SOURCE	The value of the graphical image source is not one of the supported values.	0x90030300
SAPI_ERR_UNSUPPORTED_GRAPHICAL_IMAGE_DATA	The graphical image data format is not a supported graphical image format.	0x90030301
SAPI_ERR_FAILED_TO_INSTALL_CA_CERT	Failed to install the CA certificate in the store. This may be a permissions problem, because the user needs administrative privileges to install a certificate in the machine store. When installing a CA certificate in the user store, the user must approve the operation by pressing the <b>Yes</b> button in the dialog box that opens during the installation process.	0x90030310
SAPI_ERR_ILLEGAL_STORE_TYPE	The store type ID is not supported for certificate installation.	0x90030320
SAPI_ERR_BUFFER_CONTAINS_NO_CERT	The buffer for the certificate contains no certificate data, or contains an unsupported format.	0x90030330
SAPI_ERR_FAILED_TO_PARSE_PKCS7_BLOB	Failed to parse the PKCS#7 blob. Verify that all blob bytes were sent to the function and that the length is correct.	0x90030340
SAPI_ERR_ILLEGAL_CONF_TYPE	A configuration value that was successfully set earlier in this SAPI session was set with an incorrect type.	0x90030350
SAPI_ERR_SIGNATURE_IS_NOT_VALID	The signature is not valid. The data was probably changed after it was signed.	0x90030360

<b>Error Code</b>	<b>Description</b>	<b>Numeric Value</b>
SAPI_ERR_INVISIBLE_IS_UNSUPPORTED	An invisible signature is not supported for this file type (the Word file type).	0x90030370
SAPI_ERR_REQUEST_FOR_TS_TIMED_OUT	Timeout for Time Stamping request.	0x90030371
SAPI_ERR_TS_REPLY_TOO_LONG	Time Stamping reply is too long.	0x90030372
SAPI_ERR_TS_SUBMIT_HTTP_ERROR	Connection to Time Stamping service problem.	0x90030373
SAPI_ERR_TS_SUBMIT_WININET_ERROR	Connection to Time Stamping service problem.	0x90030374
SAPI_ERR_FAILED_TO_READ_TS_URL_VAL	Missing configuration value for Time Stamping service URL.	0x90030375
SAPI_ERR_FAILED_TO_ENCODE_PKCS7_WITH_TS	Failed to encode the time stamp into the digital signature.	0x90030376
SAPI_ERR_FAILED_TO_VERIFY_TS	Failed to verify the time stamp.	0x90030377
SAPI_ERR_TS_FAILED_TO_COMPARE_RANDOM_VAL	Failed to compare random information between Time Stamping request and response.	0x90030378
SAPI_ERR_TS_FAILED_TO_COMPARE_HASH_VAL	Failed to compare hash values between the Time Stamping request and response.	0x90030379
SAPI_ERR_TS_ILLEGAL_URL_FORMAT	Illegal URL format of the Time Stamping service.	0x9003037A
SAPI_ERR_INVALID_TIMESTAMP	Invalid time stamp.	0x90030380
SAPI_ERR_TIME_WASNT_FOUND	Time was not found in the time stamp.	0x90030381
SAPI_ERR_UNSUPPORTED_DATA_TYPE	Unsupported type for the custom field.	0x90030382

<b>Error Code</b>	<b>Description</b>	<b>Numeric Value</b>
SAPI_ERR_ILLEGAL_CUSTOM_FIELD_ID	Illegal custom field ID.	0x90030383
SAPI_ERR_SIGN_HASH_CALLED_TWICE_OR_BAD_HASH_LEN	Either the continuous signing operation was called twice or the hash mechanism is not SHA_1.	0x90030384
SAPI_ERR_SIGN_HASH_CANT_RETURN_PKCS7_LEN	Problem with hash signature.	0x90030385
SAPI_ERR_SIGN_HASH_ACQUIRE_CONTEXT	A Microsoft CSP problem.	0x90030386
SAPI_ERR_SIGN_HASH_CREATE_HASH	Problem creating hash value.	0x90030387
SAPI_ERR_SIGN_HASH_CRYPT_HASH_DATA	Problem creating hash value.	0x90030388
SAPI_ERR_SIGN_HASH_CRYPT_SIGN_HASH	Problem creating hash value.	0x90030389
SAPI_ERR_AR_CSP_CANNOT_BE_FOUND	Problem using ARCSP for verification purposes.	0x9003038a
SAPI_ERR_PKCS7_BLOB_HAS_A_BAD_ALGID	Problem verifying a signature. Probably an unsupported signature algorithm is used	0x9003038b
SAPI_ERR_CERT_VALIDATE_FAILURE	Problem validating certificate.	0x9003038c
SAPI_ERR_VERIFY_HASH_ACQUIRE_CONTEXT	Problem during hash calculations.	0x9003038d
SAPI_ERR_VERIFY_HASH_CREATE_HASH	Problem during hash calculations.	0x9003038e
SAPI_ERR_VERIFY_HASH_CRYPT_HASH_DATA	Problem during hash calculations.	0x9003038f
SAPI_ERR_VERIFY_HASH_CRYPT_VERIFY_HASH	Problem during hash calculations.	0x90030390
SAPI_ERR_VERIFY_HASH_CRYPT_IMPORT_KEY	Problem during PKCS#1 signature validation.	0x90030391
SAPI_ERR_VERIFY_MISSING_CERT	Problem during PKCS#1 signature validation. A certificate is required.	0x90030392

Error Code	Description	Numeric Value
SAPI_ERR_VERIFY_BAD_HASH_CERT	Problem during signature validation when an advanced signature is required.	0x90030393
SAPI_ERR_VERIFY_BAD_ADVANCED_SIG_LEVEL	Problem during signature validation when an advanced signature is required.	0x90030394
SAPI_ERR_OK_WITH_MISMATCH_IN_FILE_VERSIONS	<p>Although the signature in an InfoPath form was successfully validated using CoSign Signature Local, there exists a file template mismatch – for example, the template was updated after the signature was generated and before the signature was validated. Note that an InfoPath template mismatch error is returned only by the <a href="#">SAPIUMExtendedLastErrorGet</a> function. An InfoPath template mismatch is not considered an error by the <a href="#">SAPISignatureFieldVerify</a> function; if the signature is valid, <a href="#">SAPISignatureFieldVerify</a> returns a success message. The only way of discovering whether there is a template mismatch is to run the <a href="#">SAPIUMExtendedLastErrorGet</a> function.</p>	0x90030395
SAPI_ERR_FAILED_TO_READ_FILE_OR_BUFFER	Problem reading file or buffer.	0x90030396
SAPI_ERR_FAILED_TO_SET_APP_NAME	Failed to set the name of the application.	0x90030397
SAPI_ERR_VERIFY_OK_FILE_CHANGE_DETECTED	Relevant for PDF files. The signature is valid, but the PDF file was changed after the document was signed.	0x90034098

Error Code	Description	Numeric Value
SAPI_ERR_FAILED_TO_CREATE_SIGNATURE_FIELD	Failed to create a signature field. This might be caused by a file permissions issue. For example, the file might have been opened by another application, or it is a read-only file.	0x900303b0
SAPI_ERR_LOCATOR_CALL_TO_INIT_MISSING	Internal error when trying to locate string markups.	0x900303a0
SAPI_ERR_LOCATOR_COULD_NOT_INIT_FILE	Failed to parse the file as a PDF file. This can occur due to a wrong file name, the file is missing, or in cases where the file is encrypted the failure can occur if the owner password and the user password are wrong.	0x900303a1
SAPI_ERR_LOCATOR_SECOND_CALL_TO_INIT_WITHOUT_FREE	Internal error when trying to locate string markups.	0x900303a2
SAPI_ERR_LOCATOR_OBJECT_CREATION_FAILED	Internal error when trying to locate string markups.	0x900303a3
SAPI_ERR_LOCATOR_INVALID_CONTEXT	Invalid context handed to the <a href="#">SAPISignatureFieldLocatorEnumInit</a> and <a href="#">SAPISignatureFieldLocatorEnumCont</a> functions.	0x900303a4
SAPI_ERR_FAILED_TO_GET_SIGNATURE_FIELD_INFO	Failed to get the signature field information. The file might be corrupt, or it was created by an unsupported version.	0x900303c0
SAPI_ERR_FAILED_TO_INIT_ENUM_SF	Failed to initialize the signature field enumeration operation.	0x900303d0
SAPI_ERR_FAILED_TO_CONT_ENUM_SF	Failed to continue the signature field enumeration operation.	0x900303e0

Error Code	Description	Numeric Value
SAPI_ERR_FAILED_TO_CLEAR_SIGNATURE_FIELD	Failed to clear a signature field. This might be due to privileges. For example, the file might have been opened by another application, or it is a read-only file.	0x900303f0
SAPI_ERR_FAILED_TO_CLEAR_SIGNATURE_FIELD_BY_POLICY	Failed to clear a signature field due to clearing policy.	0x900303f1
SAPI_ERR_FAILED_TO_GUI_GET_SIGNATURE_FIELD_DETAILS	Failed to get signature field signature information to be displayed in a GUI.	0x900303f2
SAPI_ERR_FAILED_TO_REMOVE_SIGNATURE_FIELD	Failed to remove a signature field. This might be due to privileges. For example, the file might have been opened by another application, or it is a read-only file.	0x90030400
SAPI_ERR_FAILED_TO_CONVERT_GRAPHIC_TO_B64	Failed to convert a graphical image object to base 64 format.	0x90030410
SAPI_ERR_FAILED_TO_CONVERT_ORIG_FORMAT_DOESNT_FIT	Problem converting image.	0x90030412
SAPI_ERR_FAILED_TO_GET_GRAPHIC_IMAGE	Failed to get a graphical image signature.	0x90030420
SAPI_ERR_FAILED_TO_SET_GRAPHIC_IMAGE	Failed to set a graphical image signature.	0x90030421
SAPI_ERR_ELECTRONIC_SIG_WITH_NO_GRAPHICAL_SIG	A generated electronic signature cannot be created with an empty graphical image.	0x90030422
SAPI_ERR_LIB_IS_NOT_INITIALIZED	A call to one of the CoSign Signature Local functions was made before calling <a href="#">SAPIInit</a> , or after calling <a href="#">SAPIFinalize</a> .	0x90030430

Error Code	Description	Numeric Value
SAPI_ERR_FAILED_TO_SIGN_FILE	Failed to sign a signature field in a file. This usually happens when there is no access to the private key. For example, when prompt for sign is set and an incorrect password is entered, or this is an unattended process and the password cannot be entered.	0x90030470
SAPI_ERR_FAILED_IN_IS_FILE_SIGNED	Failed to determine whether the file is signed.	0x90030480
SAPI_ERR_FILE_TYPE_NOT_SUPPORTED	The file type value that was used is not one of the supported values.	0x90030490
SAPI_ERR_FAILED_TO_VERIFY_FILE	Failed to verify a signature field in a file.	0x90030570
SAPI_ERR_INVALID_SIG_FIELD_HANDLE	The signature field handle that is used is either not initialized or is corrupt.	0x90030600
SAPI_ERR_INVALID_SIG_FIELD_ENUM_CONTEXT	The signature field enumeration operation context is either not initialized or is corrupt.	0x90030601
SAPI_ERR_VERSION_NOT_SUPPORTED	The signature field that is being processed was created by a version that is not supported.	0x90030602
SAPI_ERR_INTERNAL_ERROR	An unexpected error occurred within CoSign Signature Local.	0x90030603
SAPI_ERR_FILE_OPEN	Failed to open the file. Check the file path and the user permissions on this file.	0x90030604

Error Code	Description	Numeric Value
SAPI_ERR_FILE_CANNOT_PARSE	Failed to parse the file. The file is either corrupt, or has features that are not supported by CoSign Signature Local.	0x90030605
SAPI_ERR_SIG_FIELD_NOT_SIGNED	The signature field that needs to be verified is not signed.	0x90030606
SAPI_ERR_FILE_NO_SUCH_PAGE	The page that is specified for the new signature field does not exist in the document.	0x90030607
SAPI_ERR_TIME_CONVERT	Failed to convert the time from the file to a format that can be displayed in the signature field.	0x90030608
SAPI_ERR_FILE_IO	CoSign Signature Local failed in accessing the file after it was opened, or there is a value in the file that points to an address that does not exist.	0x90030609
SAPI_ERR_UNKNOWN_SIG_VALUE_KIND	The signature value is not in one of the supported formats. It is either a corrupt value, or the signature field that was created by a CoSign Signature Local version that is not supported by the current version.	0x9003060a
SAPI_ERR_UNSUPPORTED_SIG_DEPENDENCY_MODE	The dependency value is not in one of the supported formats. It is either a corrupt value, or the signature field was created by a CoSign Signature Local version that is not supported by the current version.	0x9003060b

Error Code	Description	Numeric Value
SAPI_ERR_UNSUPPORTED_SIG_TYPE	The signature type is a value that is not supported by CoSign Signature Local. This error usually occurs when trying to work with a signature field of a Word file that was created by the Word plug-in, and its scope of signing is not the entire file. CoSign Signature Local can only work with Word's Signature fields that are of scope entire file.	0x9003060c
SAPI_ERR_FIELD_NOT_FOUND	Failed to find the signature field in the file. Verify the signature field was not removed after its handle was retrieved.	0x9003060d
SAPI_ERR_FIELD_NOT_SIGNED	Not in use.	0x9003060e
SAPI_ERR_INVALID_FIELD_SIZE	The signature field size is invalid. The width or height is either too small or too big.	0x9003060f
SAPI_ERR_CERTIFIED_CANT_CHANGE_FILE	The PDF file is certified and therefore cannot be changed.	0x90030610
SAPI_ERR_CERTIFIED_CANT_CERTIFY_AGAIN	The PDF file is certified and therefore cannot be certified again.	0x90030611
SAPI_ERR_FAILED_TO_CREATE_FILE_HANDLE_BY_NAME	Problem creating a file handle by name.	0x90030612
SAPI_ERR_FAILED_TO_GET_FILE_MEM_DATA	Problem retrieving memory information of a file handle.	0x90030613
SAPI_ERR_FAILED_TO_GET_FILE_HANDLE	Problem getting a file handle.	0x90030614
SAPI_ERR_INVALID_HANDLE	The handle that was passed to the function is not valid.	0x90030700

Error Code	Description	Numeric Value
SAPI_ERR_INVALID_ENUM_CONTEXT	The context that was passed to the function is incorrect.	0x90030701
SAPI_ERR_INVALID_PARAM	The value of one of the parameters is invalid. It was either not initialized or is corrupt.	0x90030702
SAPI_ERR_BUFFER_TOO_SMALL	The memory allocated for the returned buffer is too small.	0x90030703
SAPI_ERR_ALREADY_EXISTS	A graphical image with the same name already exists in the user account, or a signature field with the same name already exists in the document.	0x90030704
SAPI_ERR_NOT_FOUND	An object (usually a graphical image object) with the specified ID was not found in the token.	0x90030705
SAPI_ERR_TOO_MANY_ITEMS	Too many objects found. Unable to select a default object from them (usually graphical image objects).	0x90030706
SAPI_ERR_NO_ITEMS	No objects were found (usually a graphical image object).	0x90030707
SAPI_ERR_OPERATION_CANCELLED_BY_USER	The user pressed the <b>Cancel</b> button in the Select Graphical Signature dialog box and no graphical image is selected.	0x90030708
SAPI_ERR_LOGO_ALREADY_EXIST	A logo already exists for the user.	0x90030709
SAPI_ERR_FAILED_TO_INIT_GR_SIG_ENUM	Problem with graphical signature enumeration.	0x90030750

<b>Error Code</b>	<b>Description</b>	<b>Numeric Value</b>
SAPI_ERR_FAILED_TO_CONT_GR_SIG_ENUM	Problem with graphical signature enumeration.	0x90030751
SAPI_ERR_FAILED_TO_GET_GR_SIG_INFO	Problem getting a graphical image.	0x90030752
SAPI_ERR_FAILED_TO_UPDATE_GR_SIG_INFO	Problem updating a graphical image.	0x90030753
SAPI_ERR_FAILED_TO_CREATE_GR_SIG	Problem creating a graphical image.	0x90030754
SAPI_ERR_FAILED_TO_DELETE_GR_SIG	Problem deleting a graphical image.	0x90030755
SAPI_ERR_GUI_SELECT_GR_SIG_RETURNED_NO_SIG	No graphical signature exists.	0x90030756
SAPI_ERR_FAILED_TO_AUTO_SELECT_GR_SIG	Failed in selecting a default graphical image. This error is returned when CoSign Signature Local searches for the default image internally, such as when signing a signature field for which a default image was never set.	0x90030757
SAPI_ERR_PAD_NOT_FOUND	Signature PAD is not connected or is unusable.	0x90030758
SAPI_ERR_FAILED_IN_SIGN_CEREMONY	Problem in signature ceremony.	0x90030760
SAPI_ERR_FAILED_TO_SET_SCP	Failed to update DocuSign Signature Appliance information into SCP in the directory.	0x90030800
SAPI_ALREADY_SIGNED_ERROR	The field is already signed.	0x90030801
SAPI_ERR_FAILED_GET_PARAMETER_LENGTH	Failed to get the parameter length.	0x90030854
SAPI_ERR_CERT_EMAIL_NOT_FIT_LICENSE	The certificate's email does not match the license.	0x90030a00
SAPI_ERR_FAILED_RETRIEVE_EMAIL	There was an error retrieving email from the certificate.	0x90030a01

Error Code	Description	Numeric Value
SAPI_TIFF_ALREADY_SIGNED_ERROR	The field to be signed is already signed, or a visible signature field is about to be signed while another visible signature already exists in the TIFF file.	0x90032000
SAPI_WORD_ERR_FAILED_INIT_WORD_APPLICATION	CoSign Signature Local failed to launch the Word application. The Word application is required for operations such as create and delete signature fields.	0x90033001
SAPI_WORD_ERR_FAILED_CLOSE_DOCUMENT	CoSign Signature Local failed to close the Word document.	0x90033002
SAPI_WORD_ERR_FAILED_SAVE_DOCUMENT	CoSign Signature Local failed to save the Word document. Verify that your user privileges on this file are correct and the file is not read only.	0x90033003
SAPI_WORD_ERR_FAILED_TO_GET_PROPERTY	CoSign Signature Local failed to get a document attribute from Word.	0x90033004
SAPI_WORD_ERR_FAILED_TO_SET_PROPERTY	CoSign Signature Local failed to set a document attribute in Word.	0x90033005
SAPI_WORD_ERR_FAILED_ADD_OLE_CONTROL	Failed to add the ARX ActiveX to the Word document.	0x90033006
SAPI_WORD_ERR_FAILED_GET_OBJECT_INFO	Failed to read the signature field information. The field might be corrupted.	0x90033007
SAPI_WORD_ERR_OBJECT_NOT_FOUND	Not in use.	0x90033008
SAPI_WORD_ERR_FAILED_TO_DELETE	Failed to remove a signature field from a Word document.	0x90033009

<b>Error Code</b>	<b>Description</b>	<b>Numeric Value</b>
SAPI_WORD_ERR_CORRUPTED_OBJECT	The signature field is corrupt and cannot be used or managed.	0x9003300a
SAPI_WORD_ERR_FAILED_TO_WRITE_XP_STREAM	Failed to write the office compatible signature data to the document. An Office compatible signature is one of the features one can define for a signature field when it is created.	0x9003300b
SAPI_OFFICE_ERR_UNKNOWN_APPLICATION	The class ID of the Word application is not one of the IDs that are known by CoSign Signature Local as Word application IDs.	0x9003300c
SAPI_WORD_ERR_SECTION_DOES_NOT_EXIST	CoSign Signature Local failed to find ARX signature information in the file.	0x9003300d
SAPI_WORD_ERR_MULTI_SECTIONS_IS_NOT_SUPPORTED	Not in use.	0x9003300e
SAPI_OFFICE_XP_SIG_UNSUPPORTED	Not in use.	0x9003300f
SAPI_PDF_ERR_NO_DOC_PERMISSION	The permissions of the file that is processed do not allow performing the requested operation.	0x90034001
SAPI_PDF_ERR_ENCRYPTED_DOC	The type of the file that is processed does not allow performing the requested operation.	0x90034002
SAPI_PDF_FAILED_TO_GET_SIGNED_VERSION	Failed to get the signed version of a PDF document	0x90034003
SAPI_CRL_ERR_UNABLE_TO_PARSE_CERT	Failed to parse the certificate.	0x90036001
SAPI_CRL_ERR_UNABLE_TO_FETCH_CRL	Failed to retrieve the CRL from the CA.	0x90036002
SAPI_CRL_ERR_TOO_LONG	The retrieved CRL is too long.	0x90036003

<b>Error Code</b>	<b>Description</b>	<b>Numeric Value</b>
SAPI_ERR_FAILED_TO_GENERATE_KEY	Failed to generate a key.	0x90037001
SAPI_ERR_FAILED_TO_EXPORT_KEY	Failed to get key information.	0x90037002
SAPI_ERR_FAILED_TO_OPEN_SYSTEM_STORE	Failed to access the Microsoft Certificate Store information.	0x90037003
SAPI_ERR_FAILED_TO_CREATE_CERT_CONTEXT	Failed to call the Microsoft Certificate Context function.	0x90037004
SAPI_ERR_FAILED_TO_ADD_CERTIFICATE_TO_STORE	Failed to add the certificate to the Microsoft Store.	0x90037005

## Appendix F: CoSign Signature Local – User Management Structures

This appendix describes the structures that can be passed as parameters to the SAPIUM functions.

### SAPI\_UM\_INFO\_STRUCT

SAPI\_UM\_INFO\_STRUCT is a structure that holds SAPIUM lib version information.

```
typedef struct {  
    unsigned long          MajorVersion;  
    unsigned long          MinorVersion;  
} SAPI_UM_INFO_STRUCT, *PSAPI_UM_INFO_STRUCT;
```

#### **Members**

*MajorVersion*

SAPIUM major version number.

*MinorVersion*

SAPIUM minor version number.

#### **See Also**

[SAPIUMLibInfoGet](#)

## SAPI\_UM\_USERS\_FILTER\_STRUCT

SAPI\_UM\_USERS\_FILTER\_STRUCT is a structure that holds filtering information for the user's enumeration operation.

```
typedef struct {  
    SAPI_UM_ENUM_USER_TYPE      UserType;  
    void                          *ExtendedFilter;  
} SAPI_UM_USERS_FILTER_STRUCT;
```

### **Members**

#### *UserType*

User type (User, Computer, Group).

#### *ExtendedFilter*

A pointer to extended filtering information. This value is currently not in use and should be set to NULL. For more information regarding the usage of this API, please contact ARX.

### **See Also**

[SAPIUMUsersEnumInitEx](#)

## SAPI\_UM\_GROUP\_RECORD

SAPI\_UM\_GROUP\_RECORD is a structure that holds all group information.

```
typedef struct {
    SAPI_TECH_ID           TechID;
    SAPI_WCHAR             Name[SAPIUM_MAX_GROUP_NAME_LENGTH];
    SAPI_WCHAR             Address[SAPIUM_MAX_GROUP_ADDRESS_LENGTH];
    SAPI_WCHAR             PhoneNumber[SAPIUM_PHONE_NUMBER_LENGTH];
    SAPI_WCHAR             Country[SAPIUM_COUNTRY_LEN+1];
    SAPI_WCHAR             DomainName[SAPIUM_MAX_DOMAIN_NAME_LENGTH];
    SAPI_WCHAR             OrganizationName[SAPIUM_MAX_ORGANIZATION_NAME_LENGTH];
    SAPI_WCHAR             OrganizationUnitName[SAPIUM_MAX_ORGANIZATIONAL_UNIT_LENGTH];
    SAPI_UM_ENUM_GROUP_KEY_SIZE      KeySize;
    SAPI_UM_ENUM_GROUP_STATUS_TYPE   GroupStatus;
    unsigned long           PackagesMask;
    unsigned long           FlagsMask;
} SAPI_UM_GROUP_RECORD;
```

### **Members**

#### *TechID*

The technical identity of the group.

#### *Name*

The name of the group.

#### *Address*

The contact address of the group.

#### *PhoneNumber*

The contact phone number of the group

#### *Country*

The country of the group using the two letter identification.

#### *DomainName*

The domain name of the group.

#### *OrganizationName*

The organization name of the group.

#### *OrganizationUnitName*

The organization unit name of the group.

#### *KeySize*

The generated RSA key size of the group in bytes. For the system default, use SAPI\_UM\_GROUP\_DEFAULT\_KEY. For the full list of values, refer to [SAPI\\_UM\\_ENUM\\_GROUP\\_KEY\\_SIZE](#).

#### *GroupStatus*

The status of the group. Refer to values of [SAPI\\_UM\\_ENUM\\_GROUP\\_STATUS\\_TYPE](#).

*PackagesMask*

Contact ARX for more information.

*FlagsMask*

Contact ARX for more information.

***See Also***

[SAPIUMGroupsEnumInit](#), [SAPIUMGroupGetByTechID](#)

## Appendix G: CoSign Signature Local – User Management TYPEDEFS

This appendix describes the SAPIUM type definitions that are not structures.

### SAPI\_UM\_SES\_HANDLE

The type definition of the SAPIUM session handle .

```
typedef void *SAPI_UM_SES_HANDLE;
```

### SAPI\_UM\_USR\_HANDLE

The type definition of the user handle.

```
typedef void *SAPI_UM_USR_HANDLE;
```

### SAPI\_UM\_CONTEXT

The type definition of the context for the continuous operations (e.g., users enumeration).

```
typedef unsigned char SAPI_UM_CONTEXT[UM_OPAQUE_CONTEXT_LENGTH];
```

### SAPI\_UM\_GUID

The type definition of the user GUID.

```
typedef char SAPI_UM_GUID[SAPIUM_GUID_LENGTH];
```

### SAPI\_TECH\_ID

The type definition of the technical identity of users and groups in DocuSign Signature Appliance.

```
typedef __int64 SAPI_TECH_ID;
```

### SAPI\_ENUM\_USERS\_CONTEXT

A different context, specific for users enumeration.

```
typedef unsigned char SAPI_ENUM_USERS_CONTEXT[UM_ENUM_USERS_CONTEXT_LENGTH]
```

## Appendix H: CoSign Signature Local – User Management ENUMS

This appendix describes the type definitions for the enumerated values (typedef enum {...}).

### SAPI\_UM\_ENUM\_USER\_TYPE

The user types that can be created in DocuSign Signature Appliance. For each user a key pair is generated and a certificate is issued.

```
typedef enum SAPI_UM_ENUM_USER_TYPE {  
    SAPI_UM_USER_TYPE_NONE           = 0,  
    SAPI_UM_USER_TYPE_USER           = 1,  
    SAPI_UM_USER_TYPE_GROUP          = 2,  
    SAPI_UM_USER_TYPE_COMPUTER       = 3  
} SAPI_ENUM_USER_TYPE;
```

#### ***ENUM values***

***SAPI\_UM\_USER\_TYPE\_NONE***

Not in use.

***SAPI\_UM\_USER\_TYPE\_USER***

Regular user.

***SAPI\_UM\_USER\_TYPE\_GROUP***

A group – can be used to enable multiple users to share the same signing key.

***SAPI\_UM\_USER\_TYPE\_COMPUTER***

A computer – can be used by services that run under local system privileges.

## SAPI\_UM\_ENUM\_COUNTER\_TYPE

The types of digital signatures counters for a given user account.

```
typedef enum SAPI_UM_ENUM_COUNTER_TYPE {  
    SAPI_UM_COUNTER_GENERAL          = 1,  
    SAPI_UM_COUNTER_1                = 2,  
    SAPI_UM_COUNTER_2                = 3  
} SAPI_UM_ENUM_COUNTER_TYPE;
```

### *ENUM values*

#### *SAPI\_UM\_COUNTER\_GENERAL*

The general counter of the user.

#### *SAPI\_UM\_COUNTER\_1*

Additional user counter.

#### *SAPI\_UM\_COUNTER\_2*

Additional user counter.

## SAPI\_UM\_ENUM\_USER\_LOGIN\_STATUS

The types of user login statuses. This can indicate why the end user cannot logon.

```
typedef enum SAPI_UM_ENUM_USER_LOGIN_STATUS {  
    SAPI_UM_USER_LOGIN_STATUS_NONE      = 0,  
    SAPI_UM_USER_LOGIN_STATUS_ENABLED   = 1,  
    SAPI_UM_USER_LOGIN_STATUS_DISABLED  = 2  
} SAPI_UM_ENUM_USER_LOGIN_STATUS;
```

### ***ENUM values***

*SAPI\_UM\_USER\_LOGIN\_STATUS\_NONE*

Not in use.

*SAPI\_UM\_USER\_LOGIN\_STATUS\_ENABLED*

User can login.

*SAPI\_UM\_USER\_LOGIN\_STATUS\_DISABLED*

User is currently disabled.

## SAPI\_UM\_ENUM\_USER\_ENROLLMENT\_STATUS

The types of user enrollment statuses.

```
typedef enum SAPI_UM_ENUM_USER_LOGIN_STATUS {  
    SAPI_UM_DONT_CARE_USER_ENROLLMENT_STATUS    = -1,  
    SAPI_UM_NONE_USER_ENROLLMENT_STATUS        = 0,  
    SAPI_UM_NEW_USER_ENROLLMENT_STATUS          = 1,  
    SAPI_UM_RENEW_USER_ENROLLMENT_STATUS        = 2  
} SAPI_UM_ENUM_USER_LOGIN_STATUS;
```

### ***ENUM values***

#### ***SAPI\_UM\_DONT\_CARE\_USER\_ENROLLMENT\_STATUS***

This value can be used in a filter to ignore enrollment statuses.

#### ***SAPI\_UM\_NONE\_USER\_ENROLLMENT\_STATUS***

User certificate is OK.

#### ***SAPI\_UM\_NEW\_USER\_ENROLLMENT\_STATUS***

User needs a new certificate.

#### ***SAPI\_UM\_RENEW\_USER\_ENROLLMENT\_STATUS***

User needs to renew the certificate.

## SAPI\_UM\_ENUM\_USER\_ENROLLMENT\_REASON

The types of reason for a new certificate for the user.

```
typedef enum SAPI_UM_ENUM_USER_ENROLLMENT_REASON {  
    SAPI_UM_NONE_USER_ENROLLMENT_REASON          = 0,  
    SAPI_UM_CN_CHANGED_USER_ENROLLMENT_REASON    = 1,  
    SAPI_UM_EMAIL_CHANGED_USER_ENROLLMENT_REASON = 2,  
    SAPI_UM_CERT_EXPIRE_USER_ENROLLMENT_REASON   = 4,  
    SAPI_UM_EMAIL_SENT_USER_ENROLLMENT_REASON    = 8  
} SAPI_UM_ENUM_USER_ENROLLMENT_REASON;
```

### *ENUM values*

#### *SAPI\_UM\_NONE\_USER\_ENROLLMENT\_REASON*

Not in use.

#### *SAPI\_UM\_CN\_CHANGED\_USER\_ENROLLMENT\_REASON*

The Common Name of the user was changed. The user therefore needs to re-enroll.

#### *SAPI\_UM\_EMAIL\_CHANGED\_USER\_ENROLLMENT\_REASON*

The email of the user was changed. The user therefore needs to re-enroll.

#### *SAPI\_UM\_CERT\_EXPIRE\_USER\_ENROLLMENT\_REASON*

The certificate of the user has expired. The user therefore needs to re-enroll.

#### *SAPI\_UM\_EMAIL\_SENT\_USER\_ENROLLMENT\_REASON*

Indicates an attempt was made to notify the user about the enrollment status.

## SAPI\_UM\_ENUM\_USER\_CERT\_STATUS\_TYPE

The certificate status of the user.

```
typedef enum SAPI_UM_ENUM_USER_CERT_STATUS_TYPE {  
    SAPI_UM_USER_CERT_STATUS_NONE          = 0,  
    SAPI_UM_USER_CERT_STATUS_EXIST        = 1,  
    SAPI_UM_USER_CERT_STATUS_NOT_EXIST    = 2  
} SAPI_UM_ENUM_USER_CERT_STATUS_TYPE;
```

### ***ENUM values***

*SAPI\_UM\_USER\_CERT\_STATUS\_NONE*

Not in use.

*SAPI\_UM\_USER\_CERT\_STATUS\_EXIST*

There is a certificate for the user.

*SAPI\_UM\_USER\_CERT\_STATUS\_NOT\_EXIST*

There is no certificate for the user.

## SAPI\_UM\_ENUM\_PENDING\_REQUEST\_STATUS\_TYPE

The certificate request status when using a WWV CA.

```
typedef enum SAPI_UM_ENUM_PENDING_REQUEST_STATUS_TYPE {  
    SAPI_UM_PENDING_REQUEST_STATUS_NONE          = 0,  
    SAPI_UM_PENDING_REQUEST_STATUS_SEND_CRQ      = 1,  
    SAPI_UM_PENDING_REQUEST_STATUS_APPROVE       = 2,  
    SAPI_UM_PENDING_REQUEST_STATUS_RETRIEVE_CERT = 3,  
    SAPI_UM_PENDING_REQUEST_STATUS_REVOKED       = 4,  
    SAPI_UM_PENDING_REQUEST_STATUS_ALL           = 999  
} SAPI_UM_ENUM_PENDING_REQUEST_STATUS_TYPE;
```

### ***ENUM values***

***SAPI\_UM\_PENDING\_REQUEST\_STATUS\_NONE***

Not in use.

***SAPI\_UM\_PENDING\_REQUEST\_STATUS\_SEND\_CRQ***

A certificate request was sent.

***SAPI\_UM\_PENDING\_REQUEST\_STATUS\_APPROVE***

The certificate request was approved.

***SAPI\_UM\_PENDING\_REQUEST\_STATUS\_RETRIEVE\_CERT***

The certificate is being retrieved.

***SAPI\_UM\_PENDING\_REQUEST\_STATUS\_REVOKED***

The certificate is in the process of revocation.

***SAPI\_UM\_PENDING\_REQUEST\_STATUS\_ALL***

Used to filter statuses.

## SAPI\_UM\_ENUM\_GROUP\_STATUS\_TYPE

The types of group statuses.

```
typedef enum SAPI_UM_ENUM_GROUP_STATUS_TYPE {  
    SAPI_UM_GROUP_STATUS_NONE          = 0,  
    SAPI_UM_GROUP_STATUS_ENABLED      = 1,  
    SAPI_UM_GROUP_STATUS_DISABLED     = 2  
} SAPI_UM_ENUM_GROUP_STATUS_TYPE;
```

### ***ENUM values***

*SAPI\_UM\_GROUP\_STATUS\_NONE*

Not in use.

*SAPI\_UM\_GROUP\_STATUS\_ENABLED*

Users of the group can login.

*SAPI\_UM\_GROUP\_STATUS\_DISABLED*

Users of the group cannot login.

## **SAPI\_UM\_ENUM\_GROUP\_KEY\_SIZE**

The key sizes of all users that belong to a group.

```
typedef enum SAPI_UM_ENUM_GROUP_KEY_SIZE{  
    SAPI_UM_GROUP_DEFAULT_KEY      = 0,  
    SAPI_UM_GROUP_NO_KEY           = 1,  
    SAPI_UM_GROUP_KEY_1024         = 128,  
    SAPI_UM_GROUP_KEY_2048         = 256,  
    SAPI_UM_GROUP_KEY_4096         = 512  
} SAPI_UM_ENUM_GROUP_KEY_SIZE;
```

### ***ENUM values***

#### ***SAPI\_UM\_GROUP\_DEFAULT\_KEY***

User key size is the system default.

#### ***SAPI\_UM\_GROUP\_NO\_KEY***

There will be no RSA key for the users in this group. These users will need to enroll their keys using an external CA.

#### ***SAPI\_UM\_GROUP\_KEY\_1024***

User key size is RSA 1024 bit.

#### ***SAPI\_UM\_GROUP\_KEY\_2048***

User key size is RSA 2048 bit.

#### ***SAPI\_UM\_GROUP\_KEY\_4096***

User key size is RSA 4096 bit.

**SAPI\_UM\_ENUM\_GROUPS\_ORDER\_TYPE**

Not in use.

## SAPI\_UM\_ENUM\_GROUP\_PACKAGE

To use this enum structure, contact ARX.

```
typedef enum SAPI_UM_ENUM_GROUP_PACKAGE {  
    SAPI_UM_GROUP_PACKAGE_NONE          = 0,  
    SAPI_UM_GROUP_PACKAGE_1            = 1,  
    SAPI_UM_GROUP_PACKAGE_2            = 2,  
    SAPI_UM_GROUP_PACKAGE_3            = 4,  
    SAPI_UM_GROUP_PACKAGE_4            = 8  
} SAPI_UM_ENUM_GROUP_PACKAGE;
```

## SAPI\_UM\_ENUM\_GROUP\_FLAGS

To use this enum structure, contact ARX.

```
typedef enum SAPI_UM_ENUM_GROUP_FLAGS {  
    SAPI_UM_GROUP_FLAGS_NONE = 0,  
    SAPI_UM_GROUP_FLAGS_ENFORCE_PROMPT_FOR_SIGN = 1  
} SAPI_UM_ENUM_GROUP_FLAGS;
```

## SAPI\_UM\_ENUM\_GROUP\_DELETE\_USER\_OP

To use this enum structure, contact ARX.

```
typedef enum SAPI_UM_ENUM_GROUP_DELETE_USER_OP {  
    SAPI_UM_GROUP_DELETE_USER_OP_NONE           = 0,  
    SAPI_UM_GROUP_DELETE_USER_OP_DELETE        = 1,  
    SAPI_UM_GROUP_DELETE_USER_OP_CLEAR_ACCOUNT = 2,  
    SAPI_UM_GROUP_DELETE_USER_OP_DISABLE      = 3  
} SAPI_UM_ENUM_GROUP_DELETE_USER_OP;
```

## SAPI\_UM\_ENUM\_LOGON\_FLAG\_TYPE

This flag can be used in the [SAPIUMLogonEx](#) function to specify attributes related to the administrative login.

```
typedef enum SAPI_UM_ENUM_LOGON_FLAG_TYPE {  
    SAPI_UM_LOGON_FLAG_TYPE_WRITE_OP           = 0,  
    SAPI_UM_LOGON_FLAG_TYPE_READONLY_OP        = 1,  
    SAPI_UM_LOGON_FLAG_TYPE_WRITE_NO_CACHE_OP = 2,  
    SAPI_UM_LOGON_FLAG_TYPE_NO_ADMIN           = 4  
} SAPI_UM_ENUM_LOGON_FLAG_TYPE;
```

### ***ENUM values***

#### *SAPI\_UM\_LOGON\_FLAG\_TYPE\_WRITE\_OP*

A regular user admin with permissions to update information in the appliance.

#### *SAPI\_UM\_LOGON\_FLAG\_TYPE\_READONLY\_OP*

The operation is read-only and therefore can be performed by connecting to an alternate appliance.

#### *SAPI\_UM\_LOGON\_FLAG\_TYPE\_WRITE\_NO\_CACHE\_OP*

In the case of multiple Primary appliances, the operation will use load balancing mechanisms for every connection attempt.

#### *SAPI\_UM\_LOGON\_FLAG\_TYPE\_NO\_ADMIN*

Intended for internal ARX usage.

## SAPI\_UM\_ENUM\_GROUP\_VALUE\_NAME

This enumerated type can be used for getting or updating additional attributes of a group. Most of these parameters are currently used by other components.

```
typedef enum SAPI_UM_ENUM_GROUP_VALUE_NAME{
    SAPI_UM_GROUP_VALUE_NONE                = 0x00000000,
    SAPI_UM_GROUP_VALUE_LICENSE_NUM_OF_USERS = 0x00000001,
    SAPI_UM_GROUP_VALUE_SAML_ID             = 0x00000002,
    SAPI_UM_GROUP_VALUE_CPS_OID_ID         = 0x00000003,
    SAPI_UM_GROUP_VALUE_CPS_URI_ID         = 0x00000004
} SAPI_UM_ENUM_GROUP_VALUE_NAME;
```

### ***ENUM values***

**SAPI\_UM\_GROUP\_VALUE\_NONE**

No Value.

**SAPI\_UM\_GROUP\_VALUE\_LICENSE\_NUM\_OF\_USERS**

The maximum number of users allowed in this group.

**SAPI\_UM\_GROUP\_VALUE\_SAML\_ID**

The SAML-ID identifier of this group.

**SAPI\_UM\_GROUP\_VALUE\_CPS\_OID\_ID**

The CPS OID of all certificates generated for users of this group.

**SAPI\_UM\_GROUP\_VALUE\_CPS\_URL\_ID**

The CPS URL of all certificates generated for users of this group.

## SAPI\_UM\_USERS\_FILTER\_EXT\_TYPE

This enumerated type can be used for extended user filtering when running a users' query.

```
typedef enum SAPI_UM_ENUM_USERS_FILTER_EXT_TYPE {  
    SAPI_UM_USERS_FILTER_EXT_TYPE_NONE          = 0,  
    SAPI_UM_USERS_FILTER_EXT_TYPE_GROUP_TECH_ID = 1  
} SAPI_UM_ENUM_USERS_FILTER_EXT_TYPE;
```

### ***ENUM values***

*SAPI\_UM\_ENUM\_USERS\_FILTER\_EXT\_TYPE\_NONE*

No additional filtering.

*SAPI\_UM\_ENUM\_USERS\_FILTER\_EXT\_TYPE\_GROUP\_TECH\_ID*

Retrieve users that belong to this specified group.

## Appendix I: CoSign Signature Local – User Management Constants

This appendix describes all the constants defined in SAPIUM.

Name	Meaning	Value
UM_OPAQUE_CONTEXT_LENGTH	The length of the opaque value in SAPI_UM_CONTEXT.	1024
SAPIUM_GUID_LENGTH	The length of the user GUID.	40
UM_ENUM_USERS_CONTEXT_LENGTH	The size of the context when enumerating users.	33000
SAPIUM_USER_RIGHT_NORMAL	Regular user privileges.	0x00000001
SAPIUM_USER_RIGHT_APPLIANCE_ADMIN	Appliance Administrative privileges. These privileges allow a user to enumerate all users in DocuSign Signature Appliance.	0x00000002
SAPIUM_USER_RIGHT_USER_ADMIN	User Administrative privileges. These privileges allow a user to manage the DocuSign Signature Appliance users (add, delete, sync, etc.).	0x00000004
SAPIUM_USER_RIGHT_USERS_ADMIN_FOR_NONE_DI_INST	User Administrative privileges for non Directory Independent installations. These privileges allow querying user information.	0x00000008
SAPIUM_USER_RIGHT_GROUP_ADMIN	Group Admin rights. These privileges allow managing users of a certain group.	0x00000100
SAPIUM_MAX_GROUP_NAME_LENGTH	Size of the group name in characters.	128
SAPIUM_MAX_GROUP_ADDRESS_LENGTH	Size of the contact address in characters.	128
SAPIUM_PHONE_NUMBER_LENGTH	Size of the contact phone number in characters.	64
SAPIUM_COUNTRY_LEN	Size of the country code in characters.	2
SAPIUM_MAX_DOMAIN_NAME_LENGTH	Size of the group's domain name in characters.	128
SAPIUM_MAX_ORGANIZATION_LENGTH	Size of the group's organization name in characters.	64
SAPIUM_MAX_ORGANIZATIONAL_UNIT_LENGTH	Size of the group's organization unit name in characters.	32
SAPIUM_MAX_NUMBER_OF_GROUPS_IN_CONTENT	Internal parameter.	30
SAPIUM_MAX_SAML_ID_LEN	Maximum length of the SAML ID	64

Name	Meaning	Value
SAPIUM_MAX_CPS_OBJECT_ID_LENGTH	Maximum length of the CPS OBJECT ID	64
SAPIUM_MAX_CPS_URI_LENGTH	Maximum length of the CPS URI	128
TOKEN_ID_MAX_LEN	The maximum length of the Token used for identifying the appliance used in this SAPIUM Session.	50
SAPIUM_MAX_NUMBER_OF_GROUPS_IN_C ONT	Internal parameter.	30

## Appendix J: CoSign Signature Local – User Management Error Messages

This appendix describes the error codes returned by SAPIUM.

Name	Meaning	Value
SAPI_OK	Function returned with no error.	0
SAPI_ERR_FAILED_TO_INIT	Failed to initialize the SAPI library. Can be caused by installing an old CoSign Signature Local version on a machine with a newer client.	0x90020100
SAPI_ERR_FAILED_TO_INIT_REG_VALUE	Cannot read the CoSign Signature Local installation path. The Install_path value cannot be read from the registry.	0x90020101
SAPI_ERR_FAILED_TO_INIT_REG_KEY	Cannot read the CoSign Signature Local installation path. The Install_path key cannot be found.	0x90020102
SAPI_ERR_FAILED_TO_INIT_LOAD_MODULE	Failed to initialize the SAPI library. Failed to load the SAPI DLL.	0x90020103
SAPI_ERR_FAILED_TO_FIND_SERVER	Cannot find a server	0x90020110
SAPI_ERR_USER_IS_NOT_LOGON	User is trying to perform an operation that requires authentication before calling the logon function.	0x90020120
SAPI_ERR_FAILED_TO_LOGON	Failed to logon to DocuSign Signature Appliance. Verify that the user name and password are correct.	0x90020130
SAPI_ERR_HANDLE_ALREADY_LOGGED_ON	SAPI session is active for another user. This error may also be returned when a second call to logon is attempted without a logoff. The error may also occur when a call to logon is made after the SAPI session was used to select objects under the credentials of the default user.	0x90020131
SAPI_ERR_FAILED_TO_UPDATE_CREDENTIAL	An attempt to change the user credentials failed.	0x90020150
SAPI_ERR_FAILED_TO_DELETE_USER	CoSign Signature Local failed to delete a user from DocuSign Signature Appliance. <a href="#">SAPIUMExtendedLastErrorGet</a> to retrieve the specific error code returned from DocuSign Signature Appliance.	0x90020160
SAPI_ERR_FAILED_TO_BEGIN_SYNC	Failed to initialize a synchronization process. Call <a href="#">SAPIUMExtendedLastErrorGet</a> to retrieve the specific error code returned from DocuSign Signature Appliance.	0x90020170

Name	Meaning	Value
SAPI_ERR_FAILED_TO_END_SYNC	Failed to end a synchronization process. Ending a synchronization process directs DocuSign Signature Appliance to delete users that are not in sync. When this function returns an error, the synchronization process is cancelled. Call <a href="#">SAPIUMExtendedLastErrorGet</a> to retrieve the specific error code returned from DocuSign Signature Appliance.	0x90020180
SAPI_ERR_FAILED_TO_GET_CA_INFO	Failed to get CA information. DocuSign Signature Appliance returns this error if the installed CA does not support the requested information type.	0x90020190
SAPI_ERR_FAILED_TO_LOGOFF	Failed to logoff from a SAPI session.	0x900201B0
SAPI_ERR_FAILED_TO_WRITE_USER	Failed to add or sync a user to DocuSign Signature Appliance. Call <a href="#">SAPIUMExtendedLastErrorGet</a> to retrieve the specific error code returned from DocuSign Signature Appliance.	0x900201D0
SAPI_ERR_FAILED_TO_WRITE_USER_EXIST	Failed to add a user to DocuSign Signature Appliance. A user with the same ID already exists.	0x900201D2
SAPI_ERR_FAILED_TO_WRITE_USER_EXCEED_LICENSE	Failed to add a user to DocuSign Signature Appliance because the maximum number of users provided under the license has been reached. Either delete users that are not active, or contact ARX to get a license that allows for more users.	0x900201D4
SAPI_ERR_FAILED_TO_UPDATE_USER	Failed to update the user record.	0x90020201
SAPI_ERR_FAILED_TO_RESET_COUNTER	Failed to reset the user's counter.	0x90020210
SAPI_ERR_FAILED_TO_GET_USER	Failed to retrieve user's information	0x90020215
SAPI_ERR_SESSION_IS_NULL	The session parameter cannot be NULL.	0x90020220
SAPI_ERR_INVALID_SES_HANDLE	The session handle used is either not initialized or is corrupt.	0x90020222
SAPI_ERR_INVALID_CONTEXT_HANDLE	The context handle used is either not initialized or is corrupt.	0x90020223
SAPI_ERR_NO_SUCH_USER	No such user exists in DocuSign Signature Appliance.	0x90020300
SAPI_ERR_TOKEN_NOT_SUPPORTED	DocuSign Signature Appliance is the only supported token for this operation.	0x90030100
SAPI_CANT_DELETE_ZOMBIE_USERS	Failed to delete zombie users. Zombie users are users that were not touched while synchronizing DocuSign Signature Appliance with the user directory. Zombie users should be deleted when the sync operation ends.	0x90030110
SAPI_ERR_FAILED_TO_ENUM_USERS	The users' enumeration operation failed.	0x90030120

Name	Meaning	Value
SAPI_FAILED_TO_ALLOCATE_MEMORY	A CoSign Signature Local operation required dynamic memory allocation and the allocation failed. Verify that there are no memory leaks or other resource problems.	0x90030130
SAPI_COM_FAILED_TO_ALLOCATE_MEMORY	A SAPICOM operation required dynamic memory allocation and the allocation failed. Verify that there are no memory leaks or other resource problems.	0x90030131
SAPI_ERR_FAILED_TO_GET_USER_INFO	Failed to get a user information structure. Call <a href="#">SAPIUMExtendedLastErrorGet</a> to retrieve the error code that was returned from DocuSign Signature Appliance.	0x90030140
SAPI_FUNCTION_NOT_SUPPORTED	This function is not supported, although defined by the CoSign Signature Local header.	0x90030150
SAPI_NO_MORE_USERS	The user enumeration operation finished and there are no more users to retrieve. This return code is usually not an error but an indication that all the users were already retrieved.	0x90030160
SAPI_ERR_FAILED_TO_CANCEL_SYNC	The cancellation of the synchronization operation of the DocuSign Signature Appliance database with the external user directory failed. Do not call <a href="#">SAPIUMUsersSyncEnd</a> before initializing a new synchronization operation.	0x90030170
SAPI_ERR_FAILED_TO_FINALIZE	Failed to release the SAPI library.	0x90030190
SAPI_ERR_NULL_VALUE_IS_NOT_ALLOWED	A NULL value is not allowed for one or more of the parameters.	0x900301d0
SAPI_COM_ERR_PARAMETER_LEN_IS_TOO_SHORT	The length allocated for the output parameter is too short. Usually the required length is returned in the corresponding length parameter.	0x90030211
SAPI_ERR_LIB_IS_NOT_INITIALIZED	A call to one of the CoSign Signature Local functions was made before calling <a href="#">SAPIInit</a> , or after calling <a href="#">SAPIFinalize</a> .	0x90030430
SAPI_ERR_FAILED_TO_SET_SCP	Failed to update DocuSign Signature Appliance information into SCP in the directory.	0x90030800
SAPI_ERR_ILLEGAL_DIRECTORY_KIND	The operation is invalid in this type of directory.	0x90030801
SAPI_NO_MORE_GROUPS	The group enumeration operation finished and there are no more groups to retrieve. This return code is usually not an error but an indication that all the groups were already retrieved.	0x90030802
SAPI_ERR_FAILED_TO_GET_USER_TECH_ID	Failed to get the technical identity of the user.	0x90030803

Name	Meaning	Value
SAPI_ERR_NUM_OF_GROUPS_LARGER_THEN_MAX	The number of groups in the retrieve request is larger than the maximum allowed (see the value of <a href="#">SAPIUM MAX NUMBER OF GROUPS IN CONT</a> )	0x90030804
SAPI_ERR_GROUP_NAME_MANDATORY	When creating or updating a group, no group name was specified.	0x90030805
SAPI_ERR_FAILED_TO_ADD_GROUP	Failed to create the group.	0x90030806
SAPI_ERR_FAILED_TO_UPDATE_GROUP	Failed to update the group record.	0x90030807
SAPI_ERR_FAILED_TO_GET_GROUP	Failed to retrieve group's information.	0x90030808
SAPI_ERR_FAILED_TO_SET_GROUP	Failed to set the group's status.	0x90030809
SAPI_ERR_FAILED_TO_DELETE_GROUP	Failed to delete the group.	0x9003080a
SAPI_ERR_FAILED_TO_ASSIGN_USER_TO_GROUP	Failed to assign the user to a group.	0x9003080b
SAPI_ERR_FAILED_TO_ENUM_GROUPS	The group's enumeration operation failed.	0x9003080c
SAPI_COM_ERR_PARAMETER_LEN_IS_TOO_LONG	Provided length is too long.	0x9003080d
SAPI_ERR_FAILED_TO_UPDATE_GROUP_EXT_DATA	Problem updating group extended info	0x90030811
SAPI_ERR_FAILED_TO_GET_GROUP_EXT_DATA	Problem getting group extended info	0x90030812
SAPI_ERR_GROUP_LICENCE_NUM_OF_USERS_PASS_LIMIT	Error internal to ARX	0x90030813
SAPI_ERR_GROUP_NEW_LICENSE_NUM_OF_USERS_NOT_VALID	Error internal to ARX.	0x90030814

# CoSign Signature SOAP API

---

This section includes the following topics:

[Introduction](#)

[Sign Operation](#)

[Verify Operation](#)

[User Management Operations](#)

[Using Web Services](#)

[Appendix A: CoSign Signature SOAP Structures](#)

[Appendix B: CoSign Signature SOAP Enumerations](#)

## Introduction

This chapter describes the CoSign Signature SOAP network API provided by appliances.

CoSign Signature SOAP enables developers to enhance their applications with digital signatures without requiring previous experience using public-key cryptography. CoSign Signature SOAP is an XML/SOAP network API via HTTPS. It can be used from any client, with any operating system, using any language and any platform.

CoSign Signature SOAP includes functions for signing files, verifying signatures, and managing signers. Signer management is only supported by servers.

As an alternative to the CoSign Signature SOAP API, starting from CoSign version 7.1, DocuSign Signature Appliance also supports the [CoSign Signature](#) API. This new API is easier to use than the CoSign Signature SOAP API but does not include user management capabilities.

The regular Web Services API cannot be used in Common Criteria EAL4+ deployments and FIPS 140-2 level 3 deployments.

The Web Services interface is composed of two functional families:

- Signature and validation related functions.

- User management related functions.

The Web Services approach can be integrated into any platform (such as MS Windows, Linux, Solaris), or any development environment (Java, .NET, PHP, PERL, etc.) that is able to produce a SOAP 1.2 interface to the Web Services executed inside the appliance.

There are several approaches to invoking a web service as part of the application:

- Native SOAP (Simple Object Access Protocol). The web application will generate an XML request and will receive an XML reply from the Web Service.

- This method requires more development resources than the following method.

- The Web Service publishes a WSDL (Web Services Definition Language) file. This file describes the interfaces of the Web Services as well as the parameters that need to be passed to the Web Service and must receive a reply from the Web Service.

- Many of the development environments have mechanisms for using the WSDL file to provide the developer with an easier method than the native SOAP.

The following sections provide detailed information about the web services offered by DocuSign Signature Appliance, and provide guidelines and samples in dedicated development environments to enable developers to easily use the Web Services solution.

**Note:** DocuSign Signature Appliance also provides a new Web Service interface to Adobe applications such as Adobe Reader and Adobe Acrobat. This solution enables Adobe Reader to sign PDF documents without installing any client application. For more information, refer to the chapter on Signing Adobe Acrobat Documents in the *DocuSign Signature Appliance User Guide*.

## Technical Information

The appliance provides a secure Web Services communication that is based on TLS (Transport Layer Security) with X.509 Server Authentication Binding.

### Accessing the signature and validation related function:

To access the signature and validation related function, use:

```
https://cosign:8080/SAPIWS/dss.asmx
```

or

```
https://<DNS name of your CoSign appliance>:8080/SAPIWS/dss.asmx
```

### Accessing the user management related function:

To access the user management related function, use:

```
https://cosign:8080/SAPIWS/spml.asmx
```

or

```
https://<DNS name of your CoSign appliance>:8080/SAPIWS/spml.asmx
```

### Remarks:

A special entry in the *hosts* file in the client side should be allocated to *cosign*, which will point to the IP address of DocuSign Signature Appliance in the network.

You also must install the ARX ROOT certificate, which is available in the CDROM in the *misc* directory. This enables the client to establish an SSL session with the appliance.

The SSL server key and certificate are temporary and should be replaced by the customer when moving to production. For instructions on how to install a new key and certificate for the SSL server, refer to the chapter on managing the appliance in the *DocuSign Signature Appliance Administrator Guide*.

## Signature and Validation API

The Signature and Validation interface is based on a standard developed by the OASIS Digital Signature Service Technical Committee. The standard is called “Digital Signature Service Core Protocols, Elements, and Bindings” (DSS Core) and can be accessed at:

<http://docs.oasis-open.org/dss/v1.0/oasis-dss-core-spec-v1.0-os.pdf>.

The Signature and Validation API is designed to be as compatible as possible with the Oasis DSS standard. Compatibility with the standard is required for the following reasons:

It increases the adoption of digital signatures into many applications.

It eases the integration of applications with the appliance.

However, Oasis DSS compatibility has some limitations:

The Oasis DSS standard covers functionality that does not yet exist in DocuSign Signature Appliance. Pay attention to the remarks in every function description to ensure proper usage of the Web Services functionality.

There are many functionalities in DocuSign Signature Appliance that are not yet covered by the Oasis DSS standard. The standard was artificially extended in the appropriate fashion so that these functionalities can be used.

For example, the signature operation of the standard was extended to support also the field creation mechanism. Therefore, the description of how to create a signature field inside a document is described as part of the signature operation, although this may be confusing for the reader.

The Web Services is very similar to the SAPICrypt functionality. Most of the Web Services types, constant values, and enumerated types are similar to the ones used in SAPICrypt.

Therefore, all values that are listed in the appendices of the SAPICrypt Reference Manual are relevant to Web Services as well.

The following information can be signed and verified:

Data Buffers

Hash values of data

XML data

PDF files

Office 2007/2010/2013 files that already contain signature fields

Word XP/2003 files that already contain file-based signature fields

TIFF files

InfoPath 2007/2010/2013 forms that already contain signature fields

For more information, see the following to view XML based information about this web service:

```
http://cosign:8080/SAPIWS/dss.asmx
```

or

```
http://<DNS name of your CoSign appliance>:8080/SAPIWS/dss.asmx
```

## ***Name Spaces Conventions***

The following name spaces are used in this chapter. Some of the name spaces are defined by the Oasis DSS standard. Others are defined by ARX and promoted to be a standard either as part of the Oasis DSS standard or by other standards.

<b>ID</b>	<b>URI</b>	<b>Description</b>
xmlns:s4	<a href="http://www.w3.org/2000/09/xmldsig#">http://www.w3.org/2000/09/xmldsig#</a>	XML Digital Signature definitions of W3 group
xmlns:s	<a href="http://www.w3.org/2001/XMLSchema">http://www.w3.org/2001/XMLSchema</a>	XML definitions of W3 group
xmlns:wSDL	<a href="http://schemas.xmlsoap.org/wSDL/">http://schemas.xmlsoap.org/wSDL/</a>	WSDL types
xmlns:soap12	<a href="http://schemas.xmlsoap.org/wSDL/soap12/">http://schemas.xmlsoap.org/wSDL/soap12/</a>	XML SOAP definitions
xmlns:http	<a href="http://schemas.xmlsoap.org/wSDL/http/">http://schemas.xmlsoap.org/wSDL/http/</a>	XML SOAP definitions
xmlns:soapenc	<a href="http://schemas.xmlsoap.org/soap/encoding/">http://schemas.xmlsoap.org/soap/encoding/</a>	XML SOAP definitions
xmlns:mime	<a href="http://schemas.xmlsoap.org/wSDL/mime/">http://schemas.xmlsoap.org/wSDL/mime/</a>	XML SOAP definitions
xmlns:s1	<code>urn:oasis:names:tc:dss:1.0:core:schema</code>	OASIS group definitions for signature and verification
xmlns:s2	<code>urn:oasis:names:tc:SAML:1.0:assertion</code>	OASIS group definitions for authentication
xmlns:s3	<a href="http://arx.com/SAPIWS/DSS/1.0">http://arx.com/SAPIWS/DSS/1.0</a>	ARX DSS definitions
xmlns:tns	<a href="http://arx.com/SAPIWS/DSS/1.0">http://arx.com/SAPIWS/DSS/1.0</a>	ARX DSS definitions

## ***Signature and Validation API Operations***

The CoSign Signature SOAP interface is based on two operations that enable activating all API signature related functions:

**Sign** – Performs signature related operations. All operations include a Sign Request and a Sign Response.

**Verify** – Performs the verify signature operation. All operations include a Verify Request and a Verify Response.

## Sign Operation

The Sign operation includes a large set of API functions, some of which do not directly perform a signature operation. For example, you can use this command to enumerate all graphical images of the given user.

The Sign operation of the Signature and Validation API is composed of a Sign Request sent by the user to the server, and a Sign Response received back from the server. The Sign Request indicates the type of API signature related operation that the server should perform.

The API Sign functions include:

[SAPISignatureFieldCreateEx](#) – Create a single signature field in a specific file.

[SAPISignatureFieldSignEx](#) – Sign a specific signature field in a specific file.

[SAPISignatureFieldCreateSignEx](#) – Create and sign a signature field in a file.

[SAPISignatureFieldClear](#) – Clear a signature field in a file. This means that all the cryptographic-related information is cleared as well as its visual objects, but the field and its attributes remain unchanged.

[SAPISignatureFieldRemove](#) – Remove a signature field from a file. A signature field can be removed whether it is signed or not.

[SAPIBufferSignEx](#) – Sign a data buffer and retrieve the digital signature.

[SAPICertificatesEnumInit](#) and [SAPICertificatesEnumCont](#) – Retrieve the list of user’s certificates.

[SAPISignatureFieldLocatorEnumInit](#) and [SAPISignatureFieldLocatorEnumCont](#) – Return signature fields location and information for the purpose of creating signatures based on field locator strings inside a document. The functionality is relevant for PDF files.

[SAPIGraphicSigImageEnumInit](#) and [SAPIGraphicSigImageEnumCont](#) – Retrieve the list of user’s graphical images.

[SAPIGraphicSigImageInfoGet](#) – Retrieve all the data of a graphical signature image.

[SAPIGraphicSigImageInfoCreate](#) – Create a new graphical signature image in DocuSign Signature Appliance.

[SAPIGraphicSigImageInfoUpdate](#) – Update a certain graphical signature image by changing the data of a graphical signature image or its name.

[SAPIGraphicSigImageInfoDelete](#) – Delete a certain graphical signature image.

[SAPICredentialChange](#) – Change the user’s password. The function first verifies that the old password is correct, and then stores the credentials calculated from the new password in the users database

[SAPIConfigurationValueSet](#) – Set a new API configuration value.

[SAPIConfigurationValueGet](#) – Retrieve a API configuration value.

[SAPICAInfoGetInit](#) and [SAPICAInfoGetCont](#) – Retrieve CA information.

[SAPITimeGet](#) – Retrieve time information. The returned value is in XML DateTime format.

Some examples are provided in [Using Web Services](#) to demonstrate the usage of a Sign Request.

Following is a detailed explanation of each of the parameters and the data sent as part of the Sign Request, and of each of the parameters and data received back in the Sign Response.

**Note:** The user activation function, which is relevant only for Common Criteria deployments, is not part of the CoSign Signature SOAP interface. It is available only through the CoSign Signature interface.

**Note:** The [SAPIGetTokenID](#) and [SAPISetTokenID](#) functions are not relevant for the CoSign Signature SOAP interface and therefore are not defined in this interface.

## Sign Request

A Sign Request is forwarded to the server and includes the following parameters:

Optional Inputs – See [Optional Inputs \(Sign Request\)](#).

Input Documents – See [Input Documents \(Sign Request\)](#).

RequestID – A parameter of type string. The input parameter will be sent in the Sign Response’s response. This parameter has meaning in asynchronous operation mode, which is not supported.

Profile – A parameter of type *s:AnyURI*. If this value is provided, it must contain the value `http://arx.com/SAPIWS/DSS1.0/`

### **Optional Inputs (Sign Request)**

For a detailed description of the Optional Inputs, refer to [Detailed Descriptions of a Sign Request’s Optional Inputs](#).

Parameter	Description
<a href="#">Signature Type</a>	A mandatory parameter that determines the actual operation of the Sign Request. The given URI defines the requested operation. Normally, a certain signature type value is mapped to a certain API function.
<a href="#">Claimed Identity</a>	A mandatory parameter for operations that require user identity (for example, digital signature operation). The claimed identity is used to authenticate the user that commits the signature request and also to associate the appropriate signature key to use. Other operations such as field creation or clearing a signature field do not require a claimed identity structure.
<a href="#">Key Selector</a>	Enables the client to select the certificate that should be used as part of the signature operation. This input is required in cases where more than one signing certificate exists for the signing user.
<a href="#">SAPI Sig Fields Settings</a>	Used when creating a new field inside a given document. All the parameters of the newly generated field are defined as the characteristics of the signature field. Some of the parameters have a visual impact, for example, the location of the field inside the document.

Parameter	Description
<a href="#">Flags</a>	Enables a client to set various flags during the signature operation. Depending on the API related function you use, any Flags value that is passed is forwarded to the specific API function. For more information, refer to the SAPICrypt documentation.
<a href="#">Return PDF Tail Only</a>	An optional Boolean input that modifies the server's behavior when processing PDF files. The PDF specification supports modifications to PDF files to be implemented as a concatenation of the modification data to the end of an existing file.
<a href="#">Signature Field Name</a>	A string value that indicates which field should be signed. This input is required in the cases where there is more than one signature field in the document.
<a href="#">Configuration Values</a>	A list of several configuration values which can be set through this interface and reticently kept inside the server. These configuration parameters are used during the signature operation. The values can be a string, an integer, or a binary value encoded in Base64 format. Configuration values can also be retrieved through this interface.
<a href="#">Graphic Image To Set</a>	A mandatory parameter whenever the Signature Type parameter is <i>set-graphic-image</i> . This parameter is used to set or update the user's graphical signature.
<a href="#">InfoPath Form Template</a>	In cases where the appliance is provided with an InfoPath form template, this parameter includes the whole template file encoded in Base64.
<a href="#">FieldLocatorOpeningPattern</a>	The pattern that indicates the start of the embedded field locator string.
<a href="#">FieldLocatorClosingPattern</a>	The pattern that indicates the end point of the embedded field locator string.
IncludeObject	Used only for XML files. If this parameter is specified when performing an XML signature, the created XML signature will be an enveloping one.
SignaturePlacement	Used only for XML files. If this parameter is specified when performing an XML signature, the created XML signature will be an enveloped one.

Parameter	Description
EstimateSignatureSize	Not used.

### ***Input Documents (Sign Request)***

For a detailed description of the Optional Inputs, refer to [Detailed Descriptions of a Sign Request's Input Documents](#).

Parameter	Description
Document	The document content that is used for digital signature operation or another operation such as signature field creation. In the case of data signing, the document will contain the value of the given data.
DocumentHash	Used when the input to the signature operation is the hash of the data.

### **Detailed Descriptions of a Sign Request's Optional Inputs**

The following sections provide detailed descriptions of a Sign Request's optional inputs.

#### ***Signature Type***

This parameter is mandatory and determines the actual operation of the Sign Request function.

## Format

Name	Type	Description
SignatureType	s:AnyURI	<p>The given URI defines the requested operation. One of the following URIs should be provided.</p> <ul style="list-style-type: none"><li>▪ <a href="urn:ietf:rfc:3369">urn:ietf:rfc:3369</a> – For a detached PKCS#7 signature. This parameter should be used when it is required to sign a buffer. This function is similar to the <a href="#">SAPIBufferSignEx</a> function.</li><li>▪ <a href="urn:ietf:rfc:2437:RSASSA-PKCS1-v1_5">urn:ietf:rfc:2437:RSASSA-PKCS1-v1_5</a> – For a detached PKCS#1 signature. This parameter should be used when it is required to sign a buffer. This function is similar to the <a href="#">SAPIBufferSignEx</a> function.</li><li>▪ <a href="urn:ietf:rfc:3275">urn:ietf:rfc:3275</a> – For an XML signature. This parameter should be used when it is required to sign XML data.</li><li>▪ <a href="http://arx.com/SAPIWS/DSS/1.0/signature-field-sign">http://arx.com/SAPIWS/DSS/1.0/signature-field-sign</a> – This parameter should be used if it is required to sign or resign a given signature field inside a file. This function is similar to the <a href="#">SAPISignatureFieldSignEx</a> function. The field that will be signed is selected according to the given <a href="#">Signature Field Name</a> parameter.</li><li>▪ <a href="http://arx.com/SAPIWS/DSS/1.0/signature-field-create-sign">http://arx.com/SAPIWS/DSS/1.0/signature-field-create-sign</a> – This parameter should be used if it is required to create and sign a new signature field. This function is similar to the <a href="#">SAPISignatureFieldCreateSign</a> function. The signature field is created with attributes according to the value of the <a href="#">SAPI Sig Fields Settings</a> parameter.</li></ul>

		<ul style="list-style-type: none"> <li>▪ <a href="http://arx.com/SAPIWS/DSS/1.0/signature-field-create">http://arx.com/SAPIWS/DSS/1.0/signature-field-create</a> – This parameter should be used when it is required to create a new signature field. The function is similar to the <a href="#">SAPISignatureFieldCreateSign</a> function. The signature field is created with attributes according to the value of the <a href="#">SAPI Sig Fields Settings</a> parameter.<a href="http://arx.com/SAPIWS/DSS/1.0/signature-field-clear">http://arx.com/SAPIWS/DSS/1.0/signature-field-clear</a> – This parameter should be used when it is required to clear an existing signature field. The function is similar to <a href="#">SAPISignatureFieldClear</a>.</li> <li>▪ <a href="http://arx.com/SAPIWS/DSS/1.0/signature-field-remove">http://arx.com/SAPIWS/DSS/1.0/signature-field-remove</a> – This parameter should be used when it is required to remove an existing signature field. The function is similar to <a href="#">SAPISignatureFieldRemove</a>.</li> <li>▪ <a href="http://arx.com/SAPIWS/DSS/1.0/enum-certificates">http://arx.com/SAPIWS/DSS/1.0/enum-certificates</a> – This parameter should be used when it is required to enumerate certificates available for signing. The function is similar to <a href="#">SAPICertificatesEnumInit</a> and <a href="#">SAPICertificatesEnumCont</a>.</li> <li>▪ <a href="http://arx.com/SAPIWS/DSS/1.0/enum-graphic-images">http://arx.com/SAPIWS/DSS/1.0/enum-graphic-images</a> – This parameter should be used when it is required to enumerate graphical images available for use during a signature operation. The function is similar to <a href="#">SAPIGraphicSigImageEnumInit</a>, <a href="#">SAPIGraphicSigImageEnumCont</a>, and <a href="#">SAPIGraphicSigImageInfoGet</a>.</li> <li>▪ <a href="http://arx.com/SAPIWS/DSS/1.0/set-graphic-image">http://arx.com/SAPIWS/DSS/1.0/set-graphic-image</a> – This parameter should be used when it is required to add, remove, or update a graphical signature image. The function is similar to the following functions: <a href="#">SAPIGraphicSigImageInfoCreate</a>, <a href="#">SAPIGraphicSigImageInfoUpdate</a> and <a href="#">SAPIGraphicSigImageInfoDelete</a>.</li> <li>▪ <a href="http://arx.com/SAPIWS/DSS/1.0/enum-field-locators">http://arx.com/SAPIWS/DSS/1.0/enum-field-locators</a> –Return signature fields location and additional information for enabling the application to create new signature fields inside the PDF document. The function is similar to <a href="#">SAPISignatureFieldLocatorEnumInit</a> and <a href="#">SAPISignatureFieldLocatorEnumCont</a>.</li> </ul>
--	--	--

### ***Claimed Identity***

This input is mandatory for operations that require user identity, such as a digital signature operation. The claimed identity is used to authenticate the user who commits the signature request and associate the appropriate signature key to use.

#### **Format**

<b>Name</b>	<b>Type</b>	<b>Description</b>
Name	s2:NameIdentifierType	<p>DocuSign Signature Appliance User Name and optionally the Domain Name of the user.</p> <p>The <i>Name</i> parameter contains a string value of the DocuSign Signature Appliance user name. The relevant domain name can be specified using the NameQualifier attribute.</p> <p>For example, &lt;Name NameQualifier="mydomain"&gt; user1 &lt;/Name&gt;</p>

Name	Type	Description
SupportingInfo	CoSignAuthDataType	<p>Required credentials for enabling a logon or signature operation.</p> <p>The <i>CoSignAuthdata</i> element is based on the following fields:</p> <ul style="list-style-type: none"> <li>▪ <b>LogonPassword (string)</b> – The password will be used to authenticate the user and enable tracing the appropriate signing key.</li> <li>▪ <b>LogonPasswordBinary (binary)</b> – The password will be used to authenticate the user and enable tracing the appropriate signing key. In this case, the password is given as a byte array. If this value is defined, it overrides the <b>LogonPassword (string)</b> value.</li> <li>▪ <b>NewLogonPassword (string)</b> – This new password will be used when trying to change the user’s password. Make sure you are not sending this parameter as part of a regular login operation.</li> <li>▪ <b>SignPassword (string)</b> – When a <i>prompt for sign</i> is set, a string should be passed in this field. Refer to <a href="#">SAPIBufferSignEx</a> or <a href="#">SAPISignatureFieldSignEx</a> for information on the <i>credential</i> field.</li> <li>▪ <b>NewSignPasswordBinary (binary)</b> – When a <i>prompt for sign</i> is set, a string should be passed in this field. Refer to <a href="#">SAPIBufferSignEx</a> or <a href="#">SAPISignatureFieldSignEx</a> for information on the <i>credential</i> field. In this case, the password is give as a byte array, If this value is defined, it overrides the <b>SignPassword(string)</b> value.</li> </ul>

## Key Selector

This parameter enables the client to select the certificate that should be used as part of the signature operation. This input is required in cases where more than one signing certificate exists for the signing user.

**Note:** More information about this section and its types can be found in the W3C's "XML Signature syntax and Processing" standard.

### Format

Specify either one of the following:

Name	Type	Description
KeyInfo	S4:KeyInfo	<p>The <i>Key Info</i> element enables you to select the certificate to use in the signature operation. There are several ways to select a certificate and you must specify the &lt;X509Data&gt; element as the only sub element of <i>KeyInfo</i>.</p> <p>The certificate selection methods include:</p> <ul style="list-style-type: none"><li>▪ Set the <i>X509Certificate</i> item of the <i>X509Data</i> field with the value of the certificate to use. This certificate will be selected for the signature operation. The certificate value should be provided in Base64 encoding.</li><li>▪ Set the <i>SPKIData</i> field with the Subject Key Identifier value of the certificate to use. The Subject Key Identifier should be provided in Base64 encoding.</li><li>▪ Set the <i>X509IssuerSerial</i> item of the <i>X509Data</i> field with the Issuer Name and the Certificate Serial Number. In this way, DocuSign Signature Appliance will be able to select the specific certificate having these characteristics. The provided serial number should be represented as a normal string.</li></ul> <p>Refer to <a href="#">KeyInfo</a> in <a href="#">Appendix A: CoSign Signature SOAP Structures</a> for more information.</p>
Other	s1:AnyType	Not used.

### ***SAPI Sig Fields Settings***

This element is used when creating a new field inside a given document. All the provided parameters are defined as the characteristics of the signature field. Some of the parameters have a visual impact, for example, the location of the field inside the document.

#### **Format**

<b>Name</b>	<b>Type</b>	<b>Description</b>
SAPISigFieldSettings	SAPISigFieldSettings Type	Signature field values that are passed as part of the create field command. For more information, refer to <a href="#">SAPISigFieldSettingsType</a> in <a href="#">Appendix A: CoSign Signature SOAP Structures</a> .

### ***Flags***

This parameter enables a client to set various flags during the signature operation or other API related functions that use the *flags* attribute. Depending on the invoked function, this parameter is passed as the *Flags* parameter to the function. For example, if the relevant function is [SAPISignatureFieldSignEx](#), this parameter is passed as the *Flags* parameter of the [SAPISignatureFieldSignEx](#) function.

#### **Format**

<b>Name</b>	<b>Type</b>	<b>Description</b>
Flags	Unsigned Int	This parameter enables the client to pass various flags that can be used as part of the operation.

### ***Return PDF Tail Only***

This optional input modifies the server's behavior when processing PDF files. The PDF specification supports modifications to PDF files to be implemented as a concatenation of the modification data to the end of an existing file.

When this field is set to true, the appliance returns only the data to be concatenated to the file, instead of returning the whole file. In this case, the data is returned in the [Signature Object](#) element instead of in the optional output [DocumentWithSignature](#).

#### **Format**

<b>Name</b>	<b>Type</b>	<b>Description</b>
ReturnPDFTailOnly	Boolean	–

### ***Signature Field Name***

This string value indicates which field should be used for the desired function. This input is required in cases where there is more than one signature field in the document.

#### **Format**

<b>Name</b>	<b>Type</b>	<b>Description</b>
SignatureFieldName	string	An identification of a field in a document. This identity will be used depending on the desired function.

### ***Configuration Values***

Several configuration values can be set through this interface and recently used by the server. These configuration parameters are used during the signature operation and other operations.

The values can be a string, an integer, or a binary value (Base64 encoded).

#### **Format**

A list of configuration values. For each value, its enumerated string and its value should be specified. The value can be a string, an integer, or a binary attribute.

<b>Name</b>	<b>Type</b>	<b>Description</b>
ConfValueID	s3:ConfIDEnum	The identifier of the configuration parameter. These parameters are identical to the ones that are used by SAPICrypt. For more information, refer to <a href="#">Appendix B: CoSign Signature SOAP Enumerations</a> .
StringValue	string	The value of the configuration.
IntegerValue	integer	The value of the configuration.
BinaryValue	s:base64Binary	The value of the configuration.

### ***Graphic Image To Set***

This parameter is mandatory when [Signature Type](#) is **set-graphic-image**. This parameter is used to set or update the user's graphical signature. Refer to [GraphicImageType](#) in Appendix A for more information about all the attributes of a graphical image.

If a graphical signature image with the same GraphicImageName attribute already exists, it will be updated. If a graphical signature image with the same GraphicImageName attribute already exists and the given <Graphic Image> element is empty, the graphical image will be deleted.

Note that the appliance only processes the <GraphicImage> subelement and the GraphicImageName attribute.

This field supports adding and updating a Logo, Initials, or a graphical signature, depending on the specified *ImageType* value. If no value is specified in the *ImageType* attribute, the type is defined as a graphical signature.

#### **Format**

<b>Name</b>	<b>Type</b>	<b>Description</b>
GraphicImageToSet	s:GraphicImageType	Refer to <a href="#">GraphicImageType</a> in <a href="#">Appendix A: CoSign Signature SOAP Structures</a> for a more detailed description of this type.

### ***InfoPath Form Template***

This Base64 value is the encoded InfoPath form template. Together with the provided InfoPath 2007/2010/2013 file, it enables the Web Services to perform various operations upon the form such as signing a digital signature.

#### **Format**

<b>Name</b>	<b>Type</b>	<b>Description</b>
InfoPathFormTemplate	Base64	The binary content of the InfoPath form template is encoded in Base64 format and sent as a parameter to the Web Services interface.

### ***FieldLocatorOpeningPattern and FieldLocatorClosingPattern***

These string values are the opening and closing patterns of the field locator strings that enable the Web Services to locate the signature fields inside the PDF document.

#### **Format**

<b>Name</b>	<b>Type</b>	<b>Description</b>
FieldLocatorOpeningPattern	string	The pattern that indicates the start of the embedded field locator string.
FieldLocatorClosingPattern	string	The pattern that indicates the end point of the embedded field locator string.

## **Detailed Descriptions of a Sign Request's Input Documents**

This section handles either the document content or the data to be signed.

In most cases, the full document is sent to the appliance for signature operations. In the latest CoSign Signature SOAP version, there are cases where it is also possible to send a hash value of the document's data.

### **Format**

One of the following should be sent as part of the request. Currently only Document is supported.

Name	Type	Description
Document	s1:DocumentType	<p>The document content that is used for a digital signature operation or another operation.</p> <p>Send this element with the &lt;Base64Data&gt; sub-element set to the document content, and the MimeType attribute set to the file type which must be one of the following values:</p> <ul style="list-style-type: none"> <li>▪ application/pdf for PDF files.</li> <li>▪ application/msword for DOC files.</li> <li>▪ application/vnd.openxmlformats-officedocument.wordprocessingml.document for DOCX files.</li> <li>▪ application/vnd.openxmlformats-officedocument.spreadsheetml.sheet for XLSX files.</li> <li>▪ image/tiff for TIFF files.</li> <li>▪ application/ms-infopath.xml for InfoPath 2007/2010/2013 forms.</li> <li>▪ application/octet-string for general buffer signing. Note that in this case, the <a href="#">Signature Type</a> should be urn:ietf:rfc:3369 or urn:ietf:rfc:2437:RSASSA-PKCS1-v1_5</li> </ul> <p>In the case of an XML signature:</p> <ul style="list-style-type: none"> <li>▪ Send this element with the &lt;Base64XML&gt; sub-element.</li> <li>▪ Do not send a MimeType.</li> <li>▪ Make sure that the <a href="#">Signature Type</a> is urn:ietf:rfc:3275</li> </ul>

Name	Type	Description
DocumentHash		<p>Use this parameter if you need to generate either a PKCS#1 signature or a PKCS#7 signature, based on a given hash value. Send this element with a <i>DigestMethod</i> element that includes the identity of the used hashed algorithms.</p> <p>Use one of the following values to indicate the algorithm:</p> <ul style="list-style-type: none"> <li>▪ <a href="http://www.w3.org/2000/09/xmlsig#sha1">http://www.w3.org/2000/09/xmlsig#sha1</a> for SHA-1.</li> <li>▪ <a href="http://www.w3.org/2001/04/xmlenc#sha256">http://www.w3.org/2001/04/xmlenc#sha256</a> for SHA-256.</li> <li>▪ <a href="http://www.w3.org/2001/04/xmlenc#sha384">http://www.w3.org/2001/04/xmlenc#sha384</a> for SHA-384.</li> <li>▪ <a href="http://www.w3.org/2001/04/xmlenc#sha384">http://www.w3.org/2001/04/xmlenc#sha384</a> for SHA-512.</li> </ul> <p>▪ The Base64 value of the hash should be sent using the <i>DigestValue</i> parameter.</p> <p>This method cannot be used for generating XML signatures.</p> <p><b>Note:</b> You can generate a signature based on a given SHA-1/SHA-256/SHA-384/SHA-512 based hash in a different manner. You can use the <code>urn:ietf:rfc:3369</code> or <code>urn:ietf:rfc:2437:RSASSA-PKCS1-v1_5</code> Signature Type, and provide the hash value as data. In the <i>flags</i> attribute, enter the following to direct DocuSign Signature Appliance to use the given hash value:</p> <ul style="list-style-type: none"> <li>▪ <code>0x00100400</code> (for forcing SHA-1)</li> <li>▪ <code>0x00004400</code> (for SHA-256)</li> <li>▪ <code>0x00008400</code> (for SHA-384)</li> <li>▪ <code>0x00010400</code> (for SHA-512)</li> </ul>
TransformedData		Not used.
Other	Any	Not used.

## Sign Response

A Sign Response is returned from the server and includes the following parameters:

Result – See [Result \(Sign Response\)](#).

Optional Outputs – See [Optional Outputs \(Sign Response\)](#).

Signature Object – This element, of type `s:base64Binary`, is returned by the server when the optional input [Signature Type](#) is set to `urn:ietf:rfc:3369` or `urn:ietf:rfc:2437:RSASSA-PKCS1-v1_5` (Data Signing), or the optional input [Return PDF Tail Only](#) is set to true, or when producing an enveloping XML signature.

This element always contains a Base64Signature element containing the output signature. It also contains a Type attribute with the value of the [Signature Type](#) optional input.

RequestID – A parameter of type string. Its value is identical to the value of the RequestID sent in the request.

Profile – A parameter of type `s:AnyURI`. A fixed value of `http://arx.com/SAPIWS/DSS1.0/` is sent in the reply.

### ***Result (Sign Response)***

A return value for the requested operation.

#### **Format**

Name	Type	Description
ResultMessage	<code>s:InternationalStringType</code>	A string that represents an answer to the request.
ResultMajor	<code>s:anyURI</code>	Major error codes will be returned as part of the result of the operation.
ResultMinor	<code>s:anyURI</code>	Minor error codes will be returned as part of the result of the operation.

### ***Optional Outputs (Sign Response)***

For a detailed description of the Optional Outputs, refer to [Detailed Descriptions of a Sign Response's Optional Outputs](#).

## Format

Parameter	Description
<a href="#">DocumentWithSignature</a>	This output parameter is returned by the server in cases involving signature fields inside documents, or in the case of an enveloped XML signature.
<a href="#">SAPISignedFieldInfo</a>	This output is returned when the optional input <a href="#">Signature Type</a> is <a href="http://arx.com/SAPIWS/DSS/1.0/signature-field-sign">http://arx.com/SAPIWS/DSS/1.0/signature-field-sign</a> , which corresponds to the <a href="#">SAPISignatureFieldSign</a> operation. This element corresponds to the <a href="#">SAPI_SIGNED_FIELD_INFO</a> structure.
<a href="#">SAPISigFieldSettings</a>	This output is returned when the optional input <a href="#">Signature Type</a> is <a href="http://arx.com/SAPIWS/DSS/1.0/signature-field-create">http://arx.com/SAPIWS/DSS/1.0/signature-field-create</a> , which corresponds to the <a href="#">SAPISignatureFieldCreate</a> operation. This value includes all available information of a field inside a document. This element corresponds to the <a href="#">SAPI_SIG_FIELD_SETTINGS</a> structure.
<a href="#">AvailableCertificate</a>	This output is returned when the optional input <a href="#">Signature Type</a> is <a href="http://arx.com/SAPIWS/DSS/1.0/enum-certificates">http://arx.com/SAPIWS/DSS/1.0/enum-certificates</a> , which corresponds to the <a href="#">SAPICertificatesEnumInit</a> and <a href="#">SAPICertificatesEnumCont</a> operations. The appliance may return multiple certificates, depending on the number of certificates that are assigned to the given user.
<a href="#">AvailableGraphicalSignature</a>	This output is returned when the optional input <a href="#">Signature Type</a> is <a href="http://arx.com/SAPIWS/DSS/1.0/enum-graphic-images">http://arx.com/SAPIWS/DSS/1.0/enum-graphic-images</a> , which corresponds to the <a href="#">SAPIGraphicSigImageEnumInit</a> , <a href="#">SAPIGraphicSigImageEnumCont</a> , and <a href="#">SAPIGraphicSigImageInfoGet</a> operations. The appliance may return multiple graphical signatures, depending on the number of graphical images that are assigned to the given user.
<a href="#">SAPISeveralSigFieldSettings</a>	This output is returned when the optional input <a href="#">Signature Type</a> is <a href="http://arx.com/SAPIWS/DSS/1.0/enum-field-locators">http://arx.com/SAPIWS/DSS/1.0/enum-field-locators</a> , which corresponds to the <a href="#">SAPISignatureFieldLocatorEnumInit</a> and <a href="#">SAPISignatureFieldLocatorEnumCont</a> operations. The appliance returns the relevant field information, for each of the field locators existing in the given document.

Parameter	Description
<a href="#">EncodedMessages</a>	This output is returned when the optional input <a href="#">Signature Type</a> is <a href="http://arx.com/SAPIWS/DSS/1.0/enum-field-locators">http://arx.com/SAPIWS/DSS/1.0/enum-field-locators</a> , which corresponds to the <a href="#">SAPISignatureFieldLocatorEnumInit</a> and <a href="#">SAPISignatureFieldLocatorEnumCont</a> operations. In this parameter, the exact field locator strings are replied. The appliance returns all the field locators existing in the given document.
EstimatedSignatureSize	Not used.

### Detailed Descriptions of a Sign Response's Optional Outputs

This section describes the information returned from the Sign operation. As mentioned above, this operation is used also for other API functions such as field creation inside a document, and retrieval of all the user's certificates.

#### *DocumentWithSignature*

This output parameter is returned by the server in cases involving signature fields inside documents or when producing an enveloped XML signature.

In these cases, the signed document is fully retrieved.

If the request is to perform a signature upon given data (that is, [Signature Type](#) is set to `urn:ietf:rfc:3369` or `urn:ietf:rfc:2437:RSASSA-PKCS1-v1_5`), or if the parameter [Return PDF Tail Only](#) is set to true or to performing an XML enveloping signature, then the returned signature is put inside the [Signature Object](#) parameter.

#### Format

Name	Type	Description
Document	s1:DocumentType	The document content that is used for the digital signature function or another function. The supported types include: <ul style="list-style-type: none"> <li>▪ Base64Data, used for most document types.</li> <li>▪ Base64XML, used for XML document types.</li> </ul> For more information, refer to <a href="#">Detailed Descriptions of a Sign Request's Input Documents</a> .

### ***SAPISignedFieldInfo***

This output is returned when the optional input [Signature Type](#) is <http://arx.com/SAPIWS/DSS/1.0/signature-field-sign>, which corresponds to the [SAPISignatureFieldSignEx](#) function. This returned information corresponds to the [SAPI\\_SIGNED\\_FIELD\\_INFO](#) structure.

#### **Format**

Name	Type	Description
SAPISignedFieldInfo	s3:SAPISignedFieldInfo	This type includes all the information of the newly signed field. See <a href="#">SAPISignedFieldInfoType</a> in Appendix A.

### ***SAPISigFieldSettings***

This output is returned when the optional input [Signature Type](#) is <http://arx.com/SAPIWS/DSS/1.0/signature-field-create>, which corresponds to the [SAPISignatureFieldCreate](#) operation. This value includes all available information of a field inside a document.

#### **Format**

Name	Type	Description
SAPISigFieldSettings	s3:SAPISigFieldSettingsType	This type includes all the information of the newly created field. See <a href="#">SAPISigFieldSettingsType</a> in <a href="#">Appendix A: CoSign Signature SOAP Structures</a> .

### ***AvailableCertificate***

This output is returned when the optional input [Signature Type](#) is <http://arx.com/SAPIWS/DSS/1.0/enum-certificates>, which corresponds to the [SAPICertificatesEnumInit](#) and [SAPICertificatesEnumCont](#) operations. The appliance may return multiple certificates, depending on the number of certificates that are assigned to the given user.

#### **Format**

Multiple occurrences of:

Name	Type	Description
AvailableCertificate	Base64Binary	A Base64 encoding of an X509 certificate.

### ***AvailableGraphicalSignature***

This output is returned when the optional input [Signature Type](#) is `http://arx.com/SAPIWS/DSS/1.0/enum-graphic-images`, which corresponds to the [SAPIGraphicSigImageEnumInit](#) and [SAPIGraphicSigImageEnumCont](#) operations. The appliance may return multiple graphical signatures of this parameter, depending on the number of graphical images that are assigned to the given user.

#### **Format**

Multiple occurrences of:

Name	Type	Description
AvailableGraphicSignature	s:GraphicImageType	A graphical image record is returned for every graphical image that belongs to the user. For more information of this type, refer to <a href="#">GraphicImageType</a> in <a href="#">Appendix A: CoSign Signature SOAP Structures</a> .

### ***SAPISeveralSigFieldSettings***

This output is returned when the optional input [Signature Type](#) is `http://arx.com/SAPIWS/DSS/1.0/enum-field-locators`, which corresponds to the [SAPISignatureFieldLocatorEnumInit](#) and [SAPISignatureFieldLocatorEnumCont](#) operations. The appliance returns the relevant field information, for each of the field locators existing in the given document.

#### **Format**

Multiple occurrences of:

Name	Type	Description
SAPISigFieldSettings	s3:SAPISigFieldSettingsType	This type includes all the information about the signature field to create. See <a href="#">SAPISigFieldSettingsType</a> in <a href="#">Appendix A: CoSign Signature SOAP Structures</a> .

### ***EncodedMessages***

This output is returned when the optional input [Signature Type](#) is `http://arx.com/SAPIWS/DSS/1.0/enum-field-locators`, which corresponds to the [SAPISignatureFieldLocatorEnumInit](#) and [SAPISignatureFieldLocatorEnumCont](#) operations. In this parameter, the exact field locator strings are replied. The appliance returns all the field locator strings existing in the given document.

## Detailed Descriptions of a Sign Response's Signature Object

The following sections provide detailed descriptions of a Sign Response's signature object

### *Signature Object*

This element is returned by the server when the optional input [Signature Type](#) is set to one of the following:

urn:ietf:rfc:3369 (PKCS#7 signing, or the optional input [Return PDF Tail Only](#) is set to true)

urn:ietf:rfc:2437:RSASSA-PKCS1-v1\_5 (PKCS#1 signing)

urn:ietf:rfc: 3275 (the required signature operation is to produce an enveloping XML signature)

This element always contains a Base64Signature element containing the output signature.

### **Format**

Name	Type	Description
Base64Signature	s:base64Binary	A digital signature.

## Verify Operation

The Verify operation of the Signature and Validation API is composed of a Verify Request sent by the user to the server, and a Verify Response received back from the server. The Verify Request indicates the type of API verification-related function that the server should perform.

The supported API operations are:

[SAPISignatureFieldVerify](#) – Verifies the validity of a single signature field.

[SAPIFileIsSigned](#) – Checks whether a file has at least one signed signature field.

[SAPIBufferVerifySignature](#) – Verifies a signature of a single buffer.

[SAPICAInfoGetInit](#) and [SAPICAInfoGetCont](#) – Retrieves the internal CA publication information, such as ROOT Certificate and CRL.

[SAPISignatureFieldEnumInit](#) and [SAPISignatureFieldEnumCont](#) and then [SAPISignatureFieldInfoGet](#) – Retrieve signature field information of a given document. As part of the verification operation, if no specific field name is provided, then all signature fields are retrieved including the non-signed ones.

[SAPITimeGet](#) – Retrieve time information. The returned value is in XML DateTime format.

## Verify Request

The Verify Request is forwarded to the server and includes the following parameters:

Optional Inputs – See [Optional Inputs \(Verify Request\)](#).

Input Documents – See [Input Documents \(Verify Request\)](#).

RequestID – A parameter of type string. The input parameter will be sent in the Sign Response's response. This parameter has meaning in asynchronous operation mode, which is not supported.

Profile – A parameter of type *s:AnyURI*. If a value is provided, it must contain the value `http://arx.com/SAPIWS/DSS/1.0/`

Signature Object – This element can be sent by the client only in cases of data verification, that is, where [Signature Type](#) is set to `urn:ietf:rfc:3369` or to `urn:ietf:rfc:3275` (enveloping XML signature).

This element should always contain a Base64Signature element containing the input signature to be verified.

### ***Optional Inputs (Verify Request)***

For more information on Optional Inputs, refer to [Detailed Descriptions of a Verify Request's Optional Inputs](#).

<b>Parameter</b>	<b>Description</b>
<a href="#">Signature Type</a>	A mandatory parameter that determines the actual operation of the Verify Request. The given URI defines the requested operation. For more information, refer to the Sign Request section
<a href="#">Claimed Identity</a>	Not relevant to the Verification operation.
Key Selector	Relevant only for verifying a PKCS#1 signature. Use this structure to set the signing certificate for the purpose of signature validation.
SAPI Sig Fields Settings	Not relevant to the Verification operation.
Flags	Enables a client to set various flags during the verification operation. Depending on the API related function you use, any flag value that is set is forwarded to the specific API function. For more information, refer to the SAPICrypt documentation.
Return PDF Tail Only	Not relevant to the Verification operation.
<a href="#">Signature Field Name</a>	Enables a client to select the signature field to be processed. When this input is omitted and more than one signature field exists in the document, all the fields in the document are processed.
<a href="#">Configuration Values</a>	A list of several configuration values which can be set through this interface and are reticently kept inside the API. These configuration parameters are used during the Verification operation. The values can be a string, an integer, or a binary value encoded in Base64 format.

### ***Input Documents (Verify Request)***

The Input Documents are the same as those listed in [Input Documents \(Sign Request\)](#) of the Sign Request.

## Detailed Descriptions of a Verify Request's Optional Inputs

The following sections provide detailed descriptions of a Verify Request's optional inputs.

### *Signature Type*

This parameter is mandatory and enables the appliance to differentiate between file verification, data verification, and other functionality.

#### **Format**

Name	Type	Description
SignatureType	S:AnyURI	<p>One of the following URIs should be provided.</p> <ul style="list-style-type: none"><li>▪ <a href="urn:ietf:rfc:3369">urn:ietf:rfc:3369</a> – For a detached PKCS#7 signature. This parameter should be used if it is required to verify a buffer. This function is similar to <a href="#">SAPIBufferVerifySignature</a>.</li><li>▪ <a href="urn:ietf:rfc:2437:RSASSA-PKCS1-v1_5">urn:ietf:rfc:2437:RSASSA-PKCS1-v1_5</a> – For a detached PKCS#1 signature. This parameter should be used if it is required to verify a buffer. This function is similar to the <a href="#">SAPIBufferVerifySignature</a> function.</li><li>▪ <a href="urn:ietf:rfc:3275">urn:ietf:rfc:3275</a> – For XML signatures. This parameter should be used when it is required to verify XML data.</li><li>▪ <a href="http://arx.com/SAPIWS/DSS/1.0/signature-field-verify">http://arx.com/SAPIWS/DSS/1.0/signature-field-verify</a> – This parameter should be used if it is required to verify a given signature field inside a file. This function is similar to <a href="#">SAPISignatureFieldVerify</a>. This value should be provided, even if it is only required to enumerate all fields in the document,.</li><li>▪ <a href="http://arx.com/SAPIWS/DSS/1.0/ca-info-get">http://arx.com/SAPIWS/DSS/1.0/ca-info-get</a> – This parameter should be used if it is required to get the CA information. This function is similar to <a href="#">SAPICAInfoGetInit</a> and <a href="#">SAPICAInfoGetCont</a>.</li><li>▪ <a href="http://arx.com/SAPIWS/DSS/1.0/time-get">http://arx.com/SAPIWS/DSS/1.0/time-get</a> – This parameter returns the current appliance time in XML DateTime format.</li></ul>

### ***Signature Field Name***

This parameter enables a client to select the signature field to be processed. When this input is omitted and more than one signature field exists in the document, all the fields in the document are processed.

#### **Format**

<b>Name</b>	<b>Type</b>	<b>Description</b>
SignatureFieldName	string	–

### ***Key Selector***

This parameter is relevant only for PKCS#1 signature verification. It enables the server to complete a signature verification procedure.

#### **Format**

Specify one of the following:

<b>Name</b>	<b>Type</b>	<b>Description</b>
KeyInfo	S4:KeyInfo	<p>The <i>Key Info</i> element enables you to select the certificate to use in the PKCS#1 verification operation.</p> <p>Set the <i>X509Certificate</i> item of the <i>X509Data</i> field with the value of the certificate to use. This certificate will be selected for the signature operation.</p> <p>The certificate value should be provided in Base64 encoding.</p> <p>Refer to <a href="#">KeyInfo</a> in <a href="#">Appendix A: CoSign Signature SOAP Structures</a> for more information.</p>

### ***Configuration Values***

For explanations, refer to the explanations in [Configuration Values](#).

### **Verify Response**

A Verify Response is returned from the server and includes the following parameters:

Result – See [Result \(Verify Response\)](#).

Optional Outputs – See [Optional Outputs \(Verify Response\)](#).

Signature Object – Not relevant.

RequestID – A parameter of type string. Its value is identical to the value of the RequestID sent in the request.

Profile – A parameter of type *s:AnyURI*. A fixed value of `http://arx.com/SAPIWS/DSS1.0/` is sent in the reply.

### ***Result (Verify Response)***

Refer to [Result \(Sign Response\)](#) in the Sign Response section.

Basically, a successful response to the verification call means that the signature is valid.

### ***Optional Outputs (Verify Response)***

The following data is returned in the Verify Response. For a complete description of the Optional Outputs, refer to [Detailed Descriptions of a Verify Response's Optional Outputs](#).

#### **Format**

Parameter	Description
<a href="#">SAPI Fields Info</a>	The output returned in the following cases: <ul style="list-style-type: none"><li>▪ The optional input <a href="#">Signature Type</a> is <code>http://arx.com/SAPIWS/DSS/1.0/signature-field-verify</code> (which corresponds to the signature field verification function <a href="#">SAPISignatureFieldVerify</a>).</li><li>▪ <a href="#">Signature Type</a> is <code>urn:ietf:rfc:3369</code> (which corresponds to the buffer verification function <a href="#">SAPIBufferVerifySignature</a>).</li><li>▪ <a href="#">Signature Type</a> is <code>urn:ietf:rfc:3275</code>, (which corresponds to the XML verification function <a href="#">SAPISignatureFieldVerify</a>).</li></ul>
<a href="#">CA Info</a>	Information about the CA, such as Certificate and CRL.
SAPITime	Appliance time, encoded in XML DateTime format.

## Detailed Descriptions of a Verify Response's Optional Outputs

This section describes the information returned from the Verify operation.

### *SAPI Fields Info*

This output is returned when the optional input [Signature Type](#) is <http://arx.com/SAPIWS/DSS/1.0/signature-field-verify>, which corresponds to the [SAPISignatureFieldVerify](#) operation.

Multiple occurrences of this element are returned, one for each signature field in the input document. The FieldStatus subelement is returned for signed (non-empty) signature fields.

#### **Format**

Multiple occurrences of:

Name	Type	Description
SigFieldSettings	s:SAPISigFieldSettingsType	Information regarding the signature field. Refer to <a href="#">SAPISigFieldSettingsType</a> in <a href="#">Appendix A: CoSign Signature SOAP Structures</a> .
SignedFieldInfo	s:SAPISignedFieldInfoType	Information regarding the digital signature information inside the signature field. If the field is not signed, this structure is returned with the value of IsSign=False. Refer to <a href="#">SAPISignedFieldInfoType</a> in <a href="#">Appendix A: CoSign Signature SOAP Structures</a> .
FieldStatus	s:VerificationStatusType	An enumerated type that corresponds to the signature status. Refer to <a href="#">VerificationStatusType</a> in <a href="#">Appendix A: CoSign Signature SOAP Structures</a> .

### *CA Info*

This element is returned by the server when [Signature Type](#) is <http://arx.com/SAPIWS/DSS/1.0/ca-info-get>, which corresponds to [SAPICAInfoGetInit](#) and [SAPICAInfoGetCont](#).

**Format**

Multiple occurrences of:

Name	Type	Description
CACertificate	base64Binary	A Base64 encoding of the CA Certificate.
CRL	base64Binary	The current CRL will be returned in Base64 encoding.
AIA	string	The value of the AIA parameter. The AIA should be part of every user certificate generated by DocuSign Signature Appliance.
CDP	string	The value of the CDP parameter. The CDP should be part of every user certificate generated by DocuSign Signature Appliance.

## User Management Operations

The User Management interface is based on a standard developed by the OASIS Provisioning Services Technical Committee. The standard is named *Service Provisioning Markup Language* (SPML) and can be accessed at:

<http://www.oasis-open.org/committees/download.php/3032/cs-pstc-spml-core-1.0.pdf>

The listed operations are relevant mostly to a Directory Independent environment. Some of the queries can be used also in other environments such as Microsoft Active Directory.

This chapter describes all User Management related functionality that is available as part of Web Services. In contrast to the Sign/Verify operations, every User Management operation is related to a certain SAPIUM function.

For more information see, refer to the following to view some XML-based information about this web service:

```
http://cosign:8080/SAPIWS/spml.asmx
```

or

```
http://<DNS name of your CoSign appliance>:8080/SAPIWS/spml.asmx
```

The API Sign functions include:

[SAPIUMUserAddEx1](#) and [SAPIUMUserAdd](#) – Add a new user.

[SAPIUMCredentialSet](#) – Set a password for an existing user.

[SAPIUMUserUpdate](#) – Update an existing user.

[SAPIUMUserSetLogonState](#) – Set the login state of the given user.

[SAPIUMCounterReset](#) – Reset the user's signature counters.

[SAPIUMUserAssignGroup](#) – Set the group of a given user.

[SAPIUMUserDelete](#) – Delete an existing user.

[SAPIUMUsersEnumInitEx](#) and [SAPIUMUsersEnumCont](#) – Enumerate the users inside DocuSign Signature Appliance.

[SAPIUMUserInfoGetEx](#) and [SAPIUMUserInfoGet](#) – Receive detailed user information.

[SAPIUMGroupAdd](#) – Add a new group.

[SAPIUMGroupUpdate](#) – Update an existing group.

[SAPIUMGroupSetStatusByTechID](#) – Set the state of the group to either Enabled or Disabled. If the group is disabled, all users who are members of the group cannot sign.

However, if a user is disabled, then even if his/her group is enabled, the user will nevertheless not be able to login.

[SAPIUMGroupDelete](#) – Delete an existing group.

[SAPIUMGroupsEnumInit](#) and [SAPIUMGroupsEnumCont](#) – Enumerate the groups inside DocuSign Signature Appliance.

[SAPIUMGroupGetByTechID](#) – Receive detailed group information.

The following SAPIUM functions are not supported:

[SAPIUMUsersSyncBegin](#), [SAPIUMUserSync](#), [SAPIUMUsersSyncEnd](#), [SAPIUMUsersSyncStop](#) – Synchronize the Users Database with an external Users Management Database.

[SAPIUMSCPSet](#) – Not relevant to Web Services operation.

[SAPIUMLibInfoGet](#) – Retrieve SAPIUM library versioning information.

**Note:** DocuSign Signature Appliance exposes a single SPML target which represents the user database. This is a flat table of users or groups, which means that no containers or hierarchy is supported.

## Name Spaces Conventions

The following name spaces are used in this chapter. Some of the name spaces are defined by the Oasis SPML standard.

ID	URI	Description
Xmlns:s	<a href="http://www.w3.org/2001/XMLSchema">http://www.w3.org/2001/XMLSchema</a>	XML definitions of W3 group
Xmlns:wSDL	<a href="http://schemas.xmlsoap.org/wSDL/">http://schemas.xmlsoap.org/wSDL/</a>	WSDL types
Xmlns:soap	<a href="http://schemas.xmlsoap.org/wSDL/soap/">http://schemas.xmlsoap.org/wSDL/soap/</a>	XML SOAP definitions
Xmlns:soap12	<a href="http://schemas.xmlsoap.org/wSDL/soap12/">http://schemas.xmlsoap.org/wSDL/soap12/</a>	XML SOAP12 definitions
Xmlns:http	<a href="http://schemas.xmlsoap.org/wSDL/http/">http://schemas.xmlsoap.org/wSDL/http/</a>	XML SOAP definitions
Xmlns:soapenc	<a href="http://schemas.xmlsoap.org/soap/encoding/">http://schemas.xmlsoap.org/soap/encoding/</a>	XML SOAP definitions
xmlns:mime	<a href="http://schemas.xmlsoap.org/wSDL/mime/">http://schemas.xmlsoap.org/wSDL/mime/</a>	XML SOAP definitions

xmlns:s1	urn:oasis:names:tc:SPML:2:0	OASIS SPML definitions
xmlns:s2	urn:oasis:names:tc:SPML:2:0:search	OASIS SPML definitions
xmlns:s4	urn:oasis:names:tc:SPML:2:0:password	Oasis SPML definitions
xmlns:s3	http://arx.com/SAPIWS/SPML/1.0	ARX SPML definitions
xmlns:tns	http://arx.com/SAPIWS/SPML/1.0	ARX SPML definitions

### User Management API Operations

The Web Services user management interface is based on the following operations:

**listTargets** – Retrieve general information about the Users database.

**add** – Add a new user.

**modify** – Update an existing user.

**delete** – Delete an existing user.

**setPassword** – Set the password of an existing user.

**search** – Retrieve users based on a query.

**iterate** – Iterate a search.

**lookup** – Retrieve a given user's information.

**closeIterator** – Close the iteration.

**addGroup** – Add a new group.

**modifyGroup** – Modify an existing group.

**deleteGroup** – Delete an existing group.

**searchGroup** – Retrieve all groups that match the search criteria.

**lookupGroup** – Retrieve a given group's information.

### Common User Management Parameters

The following parameters are used by the User Management API operations.

### ***RequestType Parameter***

The *RequestType* parameter must be added to each of the API operations requests. The parameter includes the following values:

<b>Name</b>	<b>Type</b>	<b>Description</b>
CoSignLogonData	S4:CoSignLogonData	The identification of the administrator who performs the operations. For more information, see <a href="#">SPML – CoSignLogonData</a> in <a href="#">Appendix A: CoSign Signature SOAP Structures</a> .
requestID	string	The request ID. This value is optional.
executionMode	string	Whether the mode of execution is <i>synchronous</i> or <i>asynchronous</i> . Only <i>synchronous</i> is supported. This value is optional.

### ***ResponseType Parameter***

The *ResponseType* parameter is received in each of the API operations responses. The parameter includes the following values:

<b>Name</b>	<b>Type</b>	<b>Description</b>
errorMessage	string	Error Message response. Several error messages can appear.
status	s1:StatusCodeType	General status code. The possible options are <i>Success</i> , <i>Failure</i> , or <i>Pending</i> .
requestID	string	The request ID that was sent in the Request.
error	s1:StatusCodeType	Specific Error status code.

### **listTargets Operations**

List all target databases inside the appliance. This operation always returns a single entity.

### ***listTargets Response***

The response returns the following parameters in addition to the [ResponseType Parameter](#).

Name	Type	Description
target	s1:TargetType	The parameter includes values that represent the Target database. In the case of DocuSign Signature Appliance, the answer is always fixed and includes a single target. For more information, see <a href="#">SPML – TargetType</a> in <a href="#">Appendix A: CoSign Signature SOAP Structures</a> .

### add Operation

Add a user to the Users database. This operation activates the [SAPIUMUserAddEx1](#) function.

### *add Request*

The request includes the following parameters in addition to the [RequestType Parameter](#).

Name	Type	Description
psoID	s1:PSOIdentifierType	The identification of the new user. For more information, see <a href="#">SPML – PSOIdentifierType</a> in <a href="#">Appendix A: CoSign Signature SOAP Structures</a> .
UserRecord	s4:UserRecord	Contains the elements of the user's record. The values of the user's records are listed in the <b>UserRecord</b> type.
TargetID	string	The ID of the target. Should be the fixed value <i>CoSignDB</i> .
returnData	s1:ReturnDataType	Directs the command what to return as the response. The value should be either <i>everything</i> or <i>data</i> .

### ***add Response***

The response returns the following parameters in addition to the [ResponseType Parameter](#).

Name	Type	Description
Pso	s1:PSOType	The information about the new user. For more information, see <a href="#">SPML – PSOType</a> in <a href="#">Appendix A: CoSign Signature SOAP Structures</a> .

### **modify Operation**

Modify a user in the Users database. This operation can activate the following API functions:

**SAPIUMUserSetLogonState** – Change the login state of the user to Enabled or Disabled. In this case, you should only supply the *LoginStatus* parameter with either the *USER\_LOGIN\_STATUS\_ENABLED* or *USER\_LOGIN\_STATUS\_DISABLED* value.

[SAPIUMUserAssignGroup](#) – Set a new group for the user. In this case, you should supply the *GroupName* parameter with the requested group name.

**SAPIUMCounterReset** – Perform a reset counter operation. To reset counter 2, specify a value of 0 in the *Counter2* field. To reset counter 3, specify a value of 0 in the *Counter3* field.

**SAPIUMUserUpdate** – Updates the user record. Only the following fields in the user record can be updated: *UserCN*, *EmailAddress*, and *RightsMask*.

### ***modify Request***

The request includes the following parameters in addition to the [RequestType Parameter](#).

Name	Type	Description
psoID	s1:PSOIdentifierType	The identification of the new user. For more information, see <a href="#">SPML – PSOIdentifierType</a> in <a href="#">Appendix A: CoSign Signature SOAP Structures</a> .
modification	s1:ModificationType	This parameter includes a single instance with an internal UserRecord sub structure that contains the updated fields. For a description of the user record fields, refer to <a href="#">SPML – UserRecord</a> .

Name	Type	Description
returnData	s1:ReturnDataType	This parameter instructs the command what to return as the response. The value should be either <i>everything</i> or <i>data</i> .

### ***modify Response***

The response returns the following parameters in addition to the [ResponseType Parameter](#).

Name	Type	Description
Pso	s1:PSOType	The information about the updated user. For more information, see <a href="#">SPML – PSOType</a> in <a href="#">Appendix A: CoSign Signature SOAP Structures</a> .

### **delete Operation**

Deletes a given user in the Users database. This operation activates the [SAPIUMUserDelete](#) function.

### ***delete Request***

The request includes the following parameters in addition to the [RequestType Parameter](#).

Name	Type	Description
psoID	s1:PSOIdentifierType	The identification of the deleted user. For more information, see <a href="#">SPML – PSOIdentifierType</a> in <a href="#">Appendix A: CoSign Signature SOAP Structures</a> .
recursive	boolean	Should be false.

### ***delete Response***

The response includes only parameters from the [ResponseType Parameter](#).

### **lookup Operation**

Retrieves a given user in the Users database. This operation activates the [SAPIUMUserInfoGetEx](#) function.

### ***lookup Request***

The request includes the following parameters in addition to the [RequestType Parameter](#).

Name	Type	Description
psoID	s1:PSOIdentifierType	The identification of the retrieved user. For more information, see <a href="#">SPML – PSOIdentifierType</a> in <a href="#">Appendix A: CoSign Signature SOAP Structures</a> .

### ***lookup Response***

The response includes the following parameters in addition to the [ResponseType Parameter](#).

Name	Type	Description
pso	s1:PSOType	Information about the retrieved user. For more information, see <a href="#">SPML – PSOType</a> in <a href="#">Appendix A: CoSign Signature SOAP Structures</a> .

### **setPassword Operation**

Set a new password for a given user in the Users database. This operation activates the [SAPIUMCredentialSet](#) function.

### ***setPassword Request***

The request includes the following parameters in addition to the [RequestType Parameter](#).

Name	Type	Description
psoID	s1:PSOIdentifierType	The identification of the updated user. For more information, see <a href="#">SPML – PSOIdentifierType</a> in <a href="#">Appendix A: CoSign Signature SOAP Structures</a> .
password	string	The requested new password.
currentPassword	string	This parameter is omitted.

### ***setPassword Response***

The response includes only parameters from the [ResponseType Parameter](#).

## search Operation

Search for users in the users database.

### *search Request*

The request includes the following parameters in addition to the [RequestType Parameter](#).

Name	Type	Description
query	s2:SearchQueryType	This parameter is not relevant. All users will be retrieved depending on the specified maxSelect value.
returnData	s1:ReturnDataType	Direct the command what to return as the response. The value should be either <i>everything</i> , <i>data</i> , or <i>identifier</i> . <ul style="list-style-type: none"><li>▪ <i>everything</i> and <i>data</i> are synonymous, and indicate all the data of the users. For more information, see <a href="#">SPML – PSOType</a>.</li><li>▪ <i>Identifier</i> indicates the User ID of the users.</li></ul>
maxSelect	int	Specify the maximum amount of returned entities. This parameter is optional.

### *search Response*

The response returns the following parameters in addition to the [ResponseType Parameter](#).

Name	Type	Description
Pso	s1:PSOType	Users' information that meets the search criteria. This parameter can have multiple instances.
iterator	s2:ResultsIteratorType	If there are more users in the query, this return value enables you to retrieve more users using the <i>iterator</i> command. An empty iterate value means all the users were queried from the appliance.

## iterate Operation

If the search command or a previous iterate command did not reply with all the users, the iterate command replies with additional users.

### *iterate Request*

The request includes the following parameters in addition to the [RequestType Parameter](#).

Name	Type	Description
iterator	s2:ResultsIteratorType	The iterator value that was sent in the reply received from previous search commands or iterator commands. Refer to <a href="#">SPML – ResultsIteratorType</a> in <a href="#">Appendix A: CoSign Signature SOAP Structures</a> for more information.

### *iterate Response*

iterateResponse is similar to [search Response](#).

## closeIterator Operation

This operation closes an iterator. This operation should be called at the end of an iterator or search command.

### *closeIterator Request*

The request includes the following parameters in addition to the [RequestType Parameter](#).

Name	Type	Description
iterator	s2:ResultsIteratorType	The iterator value that was sent in the reply received from previous search commands or iterator commands. Refer to <a href="#">SPML – ResultsIteratorType</a> in <a href="#">Appendix A: CoSign Signature SOAP Structures</a> for more information.

### *closeIterator Response*

The response only includes parameters from the [ResponseType Parameter](#).

## addGroup Operation

Add a group to the Users database. This operation activates the [SAPIUMGroupAdd](#) function.

### *addGroup Request*

The request includes the following parameters in addition to the [RequestType Parameter](#).

Name	Type	Description
psoID	s1:PSOIdentifierType	The identification of the new user. For more information, see <a href="#">SPML – PSOIdentifierType</a> in <a href="#">Appendix A: CoSign Signature SOAP Structures</a> .
GroupRecord	s4:UserRecord	Contains the elements of the group's record. For more information, see <a href="#">SPML – GroupRecord</a> .
TargetID	string	The ID of the target. This should be the fixed value <i>CoSignDB</i> .
returnData	s1:ReturnDataType	Directs the command what to return as the response. The value should be either <i>everything</i> or <i>data</i> .

### *addGroup Response*

The response returns the following parameters in addition to the [ResponseType Parameter](#).

Name	Type	Description
Pso	s1:PSOType	The information about the new group. For more information, see <a href="#">SPML – PSOType</a> in <a href="#">Appendix A: CoSign Signature SOAP Structures</a> .

## modifyGroup Operation

Modify a group in the Users database. This operation activates the [SAPIUMGroupUpdate](#) function.

### ***modifyGroup Request***

The request includes the following parameters in addition to the [RequestType Parameter](#).

Name	Type	Description
psoID	s1:PSOIdentifierType	The identification of the updated group. For more information, see <a href="#">SPML – PSOIdentifierType</a> in <a href="#">Appendix A: CoSign Signature SOAP Structures</a> .
modification	s1:ModificationType	Includes a single instance with an internal GroupRecord sub structure that contains the updated fields. For a description of the group record fields, refer to <a href="#">SPML – GroupRecord</a> .
returnData	s1:ReturnDataType	Instructs the command what to return as the response. The value should be either <i>everything</i> or <i>data</i> .

### ***modifyGroup Response***

The response returns the following parameters in addition to the [ResponseType Parameter](#).

Name	Type	Description
Pso	s1:PSOType	The information about the updated group. For more information, see <a href="#">SPML – PSOType</a> in <a href="#">Appendix A: CoSign Signature SOAP Structures</a> .

### **deleteGroup Operation**

Deletes a given group in the database. This operation activates the [SAPIUMGroupDelete](#) [SAPIUMGroupDelete](#) function.

### ***deleteGroup Request***

The request includes the following parameters in addition to the [RequestType Parameter](#).

Name	Type	Description
psoID	s1:PSOIdentifierType	The identification of the deleted group. For more information, see <a href="#">SPML – PSOIdentifierType</a> in <a href="#">Appendix A: CoSign Signature SOAP Structures</a> .
recursive	boolean	Should be false.
Group_user_op	integer	The parameter value should be 0. For more information, contact ARX.

### ***deleteGroup Response***

The response includes only parameters from the [ResponseType Parameter](#).

### **lookupGroup Operation**

Retrieves a given group in the database. This operation activates the [SAPIUMGroupGetByTechID](#) function.

### ***lookupGroup Request***

The request includes the following parameters in addition to the [RequestType Parameter](#).

Name	Type	Description
psoID	s1:PSOIdentifierType	The identification of the group. For more information, see <a href="#">SPML – PSOIdentifierType</a> in <a href="#">Appendix A: CoSign Signature SOAP Structures</a> .

### ***lookupGroup Response***

The response includes the following parameters in addition to the [ResponseType Parameter](#).

Name	Type	Description
pso	s1:PSOType	Information about the retrieved group. For more information, see <a href="#">SPML – PSOType</a> in <a href="#">Appendix A: CoSign Signature SOAP Structures</a> .

## searchGroup Operation

Search for groups in the database.

### *searchGroup Request*

The request includes the following parameters in addition to the [RequestType Parameter](#).

Name	Type	Description
returnData	s1:ReturnDataType	Direct the command what to return as the response. The value should be either <i>everything</i> , <i>data</i> , or <i>identifier</i> . <ul style="list-style-type: none"><li>▪ <i>everything</i> and <i>data</i> are synonymous, and indicate all the data of the groups. For more information, see <a href="#">SPML – GroupRecord</a>.</li><li>▪ <i>Identifier</i> indicates the GroupName of the groups.</li></ul>
maxSelect	int	Specify the maximum amount of returned entities. This parameter is optional.

### *searchGroup Response*

The response returns the following parameters in addition to the [ResponseType Parameter](#).

Name	Type	Description
pso	s1:PSOType	Groups information that meets the search criteria. This parameter can have multiple instances.

**Note:** There is no iteration function for groups. Therefore, if maxSelect is included in the request and its value is smaller than the total number of groups in the database, the remaining groups will not be retrieved. It is therefore advisable not to include the maxSelect parameter in the request.

## Using Web Services

This chapter provides information on how to activate the Web Services from a variety of client development environments. A code sample is provided for every discussed environment, showing how to use the appliance to sign a buffer.

For each development environment, there are tools that use the Web Services WSDL file to either generate code or define methods that enable the application to easily access the Web Services. This chapter also describes these tools.

### Java

In the Java environment there is a utility called Axis which can be downloaded from <http://axis.apache.org/axis/>. Activating this tool generates classes in Java that can access the Web Services implementation in DocuSign Signature Appliance.

### PHP

The following sample is based on PHP5 and demonstrates how to use PHP for PDF signing. Other types of documents can be signed in a very similar manner.

```
<?php
try
{
    $ch = curl_init();
    curl_setopt($ch, CURLOPT_SSL_VERIFYPEER, false);

    // initiating the SOAP client side based on the given WSDL file
    $wsdl_url = "https://cosign:8080/sapiws/dss.asmx?wsdl";
    $client = new SoapClient($wsdl_url);

    // this namespace should be defined in several places
    $ns = "http://www.w3.org/2001/XMLSchema";

    // Authentication. In the case of Active Directory, the domain name should
    // be defined in the NameQualifier attribute
    $req->OptionalInputs->ClaimedIdentity->Name = "JMiller";
    $req->OptionalInputs->ClaimedIdentity->SupportingInfo->LogonPassword =
        "12345678";

    // Signature Type
    $req->OptionalInputs->SignatureType =
        "http://arx.com/SAPIWS/DSS/1.0/signature-field-sign";

    // Data
    // setting the GMT offset of the signature operation
    $confs->ConfValue[0]->ConfValueID = "GMTOffset";
}
```

```

$confs->ConfValue[0]->IntegerValue ="60" ;

$req->OptionalInputs->ConfigurationValues=$confs;

// reading the PDF file
$req->InputDocuments = array(
    'Document' => array(
        'Base64Data' => array(
            '_' => file_get_contents('aaa.pdf'),
            'MimeType' => 'application/pdf'
        )
    )
);

// sending the request
$dssReq->SignRequest = $req;
$output = $client->DssSign($dssReq);

echo "result is: ";
echo $output->DssSignResult->Result->ResultMajor;
echo "\n";
echo $output->DssSignResult->Result->ResultMinor;
echo "\n";

if ( $output->DssSignResult->Result->ResultMajor !=
    "urn:oasis:names:tc:dss:1.0:resultmajor:Success")
{
    echo $output->DssSignResult->Result->ResultMessage->_ ;
    echo "\n\n";
}
else
{
    $value =
        $output->DssSignResult->OptionalOutputs->DocumentWithSignature->Document->Base64Data;

    // write the signed pdf file
    file_put_contents('bbb.pdf', $value->_);
}
}
catch (Exception $e)
{
    echo "Error!<br />";
    echo $e -> getMessage ();
}

```

## Appendix A: CoSign Signature SOAP Structures

The following structures are defined and used as part of the CoSign Signature SOAP interface.

### KeyInfo

This structure is used to manage information of a key and a certificate. This information can either be extracted from the appliance by requesting a list of all user's certificates, or directed into the appliance when defining with which key and certificate to sign.

#### Format

Member	Type	Description
X509Data	s4:X509DataType	X509 Certificate value. Refer to <a href="#">X509DataType</a> for a more detailed description of how this structure is defined.
PGPData	s4:PGPDataType	Not used.
RetrievalMethod	s4:RetrievalMethodType	Not used.
SPKIData	s4:SPKIDataType	Not used.
KeyName	string	Not used.
KeyValue	KeyValueTypes	Not used.
MgmtData	string	Not used

### X509DataType

This structure is used to manage X509 certificate information and other X509 data such as the CRL. This structure is used in a variety of cases, such as an enumeration of user certificates, and also when retrieving the Certificate or CRL.

#### Format

Member	Type	Description
X509Certificate	Base64Binary	Base64 encoding of an X509 certificate.
X509IssuerSerial	s4:X509IssuerSerialType	Information about the serial number and issuer name of the X509 certificate. The elements include: <ul style="list-style-type: none"><li>509IssuerName (string) – The name of the issuer who approved the certificate. The Common-Name field should be supplied.</li><li>X509SerialNumber – The serial number of the specific certificate.</li></ul>
X509CRL	Base64Binary	A representation of a CRL. Not used.
X509SKI	Base64Binary	Base64 encoding of the Subject key identifier field inside the X509 Certificate.

Member	Type	Description
X509SubjectName	string	Not used.

### TimeDateFormatType

This format defines the displayed time and date formats.

#### Format

Member	Type	Description
ExtTimeFormat	s3:ExtendedTimeFormatEnum	An enumerated value that specifies whether to display GMT related information in addition to the displayed date and time. Refer to <a href="#">Appendix B: CoSign Signature SOAP Enumerations</a> .
DateFormat	string	The display format of the date (i.e., “dd mmm yyyy”).
TimeFormat	string	The display format of the time (i.e., “hh:mm”).

### GraphicImageType

This structure defines the graphical signature image that can be sent to the appliance or retrieved from the appliance.

#### Format

Member	Type	Description
GraphicImage	base64Binary	The actual data of the graphical image.
GraphicImageName	string	The ID of the graphical image.
DataFormat	Unsigned int	The format of the graphical image. Refer to <a href="#">GraphicImageFormatEnum</a> .
ImageConvertedFormat	GraphicImageFormatEnum	The output graphical image might be converted from the original format in which the graphical image is stored. This field contains the type of the converted graphical image. Refer to <a href="#">GraphicImageFormatEnum</a> for more information.
Height	int	The height of the image.
Width	int	The width of the image.
ImageType	s3:GraphicImageTypeEnum	The type of image. Refer to <a href="#">GraphicImageTypeEnum</a> .

### SAPISignedFieldInfoType

This structure defines all information of a signed signature field.

## Format

Member	Type	Description
Certificate	base64Binary	The relevant certificate that was used as part of the signature operation.
GraphicImage	s3:GraphicImage Type	The graphical image of the digital signature operation.
GraphicLogo	s3:GraphicImage Type	The logo image of the digital signature operation.
SignerName	string	The user who signed the document.
IsSigned	Boolean	Whether the field is signed.
SignatureTime	dateTime	The date and time of the signature operation.
Reason	string	The reason for the signature operation.
DependencyString	string	An indication of the dependency mode of the signed field. Refer to <a href="#">DependencyModeEnum</a> for the list of available strings.

## SAPISigFieldSettingsType

This structure is used when creating a new field inside a given document or retrieving signature field information from a file. All the parameters represent characteristics of the signature field. Some of the parameters have a visual impact, such as X and Y, which determine the location of the field inside the document.

## Format

Member	Type	Description
TimeFormat	s3:TimeDate FormatType	The format of the time displayed in the signature.
ExtendedInfo	dss:AnyType	Additional information about the signature field. For details contact ARX.
Name	string	The ID of the field.
DependencyMode	s3:Dependency ModeEnum	An enumerated type that specifies the dependency mode of the field. Refer to <a href="#">DependencyModeEnum</a> .
Signature Type	s3:SignatureType Enum	An enumerated type that specifies the type of signature. Refer to <a href="#">SignatureTypeEnum</a> .
Page	int	The number of the page in which the signature field is located.
X	int	The X position of the signature field.

Member	Type	Description
Y	int	The Y position of the signature field.
Height	int	The height of the signature field.
Width	int	The width of the signature field.
AppearanceMask	Unsigned int	An appearance mask that specifies which information is displayed inside the digital signature. For more information, refer to the description of <a href="#">SAPI_SIG_FIELD_SETTINGS</a> in <a href="#">Appendix A: CoSign Signature Local – Signing/Verifying Structures</a> in the SAPICrypt section.
LabelsMask	Unsigned int	A mask that specifies which fields are displayed with their caption. For more information, refer to the description of <a href="#">SAPI_SIG_FIELD_SETTINGS</a> in <a href="#">Appendix A: CoSign Signature Local – Signing/Verifying Structures</a> in the SAPICrypt section.
EmptyFieldLabel	string	A string that is displayed inside the empty signature field.
Invisible	Boolean	Indicates whether the new field is a visible or invisible signature field. TRUE means that the field is invisible.
Flags	unsigned int	Flags included in the signature field. For more information, refer to the descriptions of the <a href="#">SAPISignatureFieldCreateEx</a> function and the  <a href="#">SAPISignatureFieldCreateSignEx</a> function in the CoSign Signature Local documentation.

## VerificationStatusType

This structure is returned for a verification request upon a signature field or a data buffer.

### Format

Member	Type	Description
SignatureStatus	int	The signature status.
CertificateStatus	s3:CertStatusEnum	Refer to the <a href="#">UserCertStatusEnum</a> enumerated type for available options.
OSCertificateStatus	Unsigned int	The certificate status as was returned by the system.

## SPML – TargetType

This structure returns information about the users database target. DocuSign Signature Appliance includes a single target.

### Format

Member	Type	Description
TargetID	string	The value 'CoSignDB'.
profile	s:anyURI	This parameter should not be included. It is listed for compatibility with the standard.
Schema	s1:SchemaType	This parameter should not be included. It is listed for compatibility with the standard.
capabilities	s1:CapabilitiesListType	This parameter should not be included. It is listed for compatibility with the standard.

## SPML – PSOIdentifierType

This structure returns information about the User Identity.

### Format

Member	Type	Description
ID	string	The ID of the User or Group.
ContainerID	s1:PSOIdentifierType	This parameter should not be included. It is listed for compatibility with the standard.
TargetID	string	The parameter that was returned in the ListTargets operation.

## SPML – CoSignLogonData

This structure returns information about the Logged On user.

### Format

Member	Type	Description
User	string	The login name of the user.
Password	string	The password of the given user.
Domain	string	The domain of the given user.

## SPML – UserRecord

This structure returns information about the User Identity.

### Format

Member	Type	Description
UserLoginName	string	The Login Name of the User.
Password	string	The password of the given user.
UserCN	string	The common name of the given user. This field can be updated in a modify operation.
EmailAddress	string	The email address of the given user. This field can be updated in a modify operation.
RightsMask	Unsigned Int	A bit mask value of the user's rights. See <a href="#">Appendix I: CoSign Signature Local – User Management Constants</a> for possible values. This field can be updated in a modify operation.
UserKind	s4:UserKindEnum	The type of user. The value can be one of the following: <i>None</i> , <i>User</i> , <i>Group</i> , or <i>Computer</i> . In most cases, the accepted value is <i>User</i> .
UpdateTime	dateTime	The last update time of the user.
Guid	string	The Guid of the user.
EnrollmentStatus	s3:EnrollmentStatusEnum	The enrollment status of the user (refer to <a href="#">EnrollmentStatusEnum</a> ).
EnrollmentReason	Unsigned Int	An enrollment reason, as specified in the API user record.
LoginStatus	S3:UserLoginEnum	This status is defined according to the <a href="#">SAPI UM ENUM USER LOGIN STATUS</a> . Currently, it indicates whether the user is enabled or disabled.
Counter1	Unsigned Int	The general counter of the user. This counter cannot be reset in a reset counter operation.
Counter2	Unsigned Int	The first resettable counter of the user.
Counter3	Unsigned Int	The second resettable counter of the user.
UserCertStatus	s3:UserCertStatusEnum	The User Certificate Status (refer to <a href="#">UserCertStatusEnum</a> ).
CertRequestStatus	s3:PendingRequestStatusEnum	The User Certificate Request Status (refer to <a href="#">PendingRequestStatusEnum</a> ).
GroupName	string	The group to which the user belongs.

## SPML – GroupRecord

This structure returns information about the Group Identity.

### Format

Member	Type	Description
GroupName	string	The Name of the Group.
Address	string	The contact address of the group.
PhoneNumber	string	The phone number of the group's contact.
Country	string	The country code of the group.
DomainName	string	The domain name of the group.
OrganizationName	string	The organization name of the group.
OrganizationUnit Name	string	The organizational unit name of the group.
KeySize	s3:GroupKeySizeEnum	The key size, for all users who are members of the group. To leave the key size of all members unchanged, specify a value of GROUP_DEFAULT_KEY. For the full list of values, refer to <a href="#">SAPI UM ENUM GROUP KEY SIZE</a> .
GroupStatus	s3:GroupStatusEnum	The login status of the group (enabled or disabled).
PackagesMask	unsignedInt	For information about this parameter, contact ARX.
FlagsMask	unsignedInt	For information about this parameter, contact ARX.

## SPML – PSOType

This structure returns user information.

### Format

Member	Type	Description
psoID	s1:PSOIdentifierType	The ID of the user. For more information, see <a href="#">SPML – PSOIdentifierType</a> .
UserRecord	s4:UserRecord	The User Record. For more information, see <a href="#">SPML – UserRecord</a> .
GroupRecord	s4:GroupRecord	The Group Record. For more information, see <a href="#">SPML – GroupRecord</a> .

## SPML – ResultsIteratorType

This structure is used as part of the Users iterator if there are more users to retrieve from the Users database.

### Format

Member	Type	Description
ID	string	The ID of the Iterator. This value should be used as opaque data.

## Appendix B: CoSign Signature SOAP Enumerations

This appendix provides descriptions of the enumeration types of the CoSign Signature SOAP interface. Many of the following enumerated values are similar to the enumerated values in the SAPICrypt documentation. For each of the enumerated types, the corresponding SAPICrypt enumerated type is listed.

### DependencyModeEnum

This type is equivalent to [SAPI\\_ENUM\\_DEPENDENCY\\_MODE](#). This enumerated type lists the modes of dependency available for a digital signature:

- None** – The dependency mode is not relevant.
- Dependent** – The signature field is dependent on other fields.
- Independent** – The signature field is not dependent on other fields.

### SignatureTypeEnum

This type is equivalent to [SAPI\\_ENUM\\_SIGNATURE\\_TYPE](#). This enumerated type lists the modes of dependency available for a digital signature:

- None** – Undefined.
- Digital** – A digital signature.
- EHash** – An electronic signature.

### GraphicImageTypeEnum

This enumerated type lists the type of graphical image available for a digital signature:

- None** – Undefined.
- GraphicImage** – A graphical signature.
- Initials** – Initials image.
- Logo** – Logo image.

### ConfIDEnum

This type is equivalent to [SAPI\\_ENUM\\_CONF\\_ID](#). This enumerated type lists all the existing configuration parameters that can be sent to the API and controls the behavior of the CoSign Signature SOAPinterface.

The following table lists all the ConfIDEnum values and their CoSign Signature Local equivalents:

ConfIDEnum Value	CoSign Signature Local Equivalent	Parameter Type
None	SAPI_ENUM_CONF_ID_NONE	
Reason	SAPI_ENUM_CONF_ID_REASON	StringValue
ChkCrIEnum	SAPI_ENUM_CONF_ID_CHK_CRL_ENUM	IntegerValue
ChkCrIVerify	SAPI_ENUM_CONF_ID_CHK_CRL_VERIFY	IntegerValue
VerifyCertSign	SAPI_ENUM_CONF_ID_VERIFY_CERT_SIGN	IntegerValue

<b>ConfIDEnum Value</b>	<b>CoSign Signature Local Equivalent</b>	<b>Parameter Type</b>
WordSFFunc	SAPI_ENUM_CONF_ID_WORD_SF_FUNC	IntegerValue
ReasonLabel	SAPI_ENUM_CONF_ID_REASON_LABEL	StringValue
DateLabel	SAPI_ENUM_CONF_ID_DATE_LABEL	StringValue
SignerLabel	SAPI_ENUM_CONF_ID_SIGNER_LABEL	StringValue
CertChainFlags	SAPI_ENUM_CONF_ID_CERT_CHAIN_FLAGS	IntegerValue
GRSigPrefName	SAPI_ENUM_CONF_ID_GR_SIG_PREF_NAME	StringValue
PDFOwnerPwd	SAPI_ENUM_CONF_ID_PDF_OWNER_PWD	StringValue
PDFUserPwd	SAPI_ENUM_CONF_ID_PDF_USER_PWD	StringValue
PDFSFFunc	SAPI_ENUM_CONF_ID_PDF_SF_FUNC	IntegerValue
GMTOffset	SAPI_ENUM_CONF_ID_GMT_OFFSET	IntegerValue
UseTimestamp	SAPI_ENUM_CONF_ID_SECURED_TS_ENABLE	IntegerValue
TimestampURL	SAPI_ENUM_CONF_ID_SECURED_TS_URL	StringValue
TimestampUser	SAPI_ENUM_CONF_ID_SECURED_TS_USER	StringValue
TimestampPWD	SAPI_ENUM_CONF_ID_SECURED_TS_PWD	StringValue
TitleValue	SAPI_ENUM_CONF_ID_TITLE	StringValue
PDFRoamingIDSrv	SAPI_ENUM_CONF_ID_PDF_ROAMING_ID_SRV	StringValue
LogoPrefName	SAPI_ENUM_CONF_ID_LOGO_PREF_NAME	StringValue
UseOCSP	SAPI_ENUM_CONF_ID_EMBED_OCSP	IntegerValue
OCSPURL	SAPI_ENUM_CONF_ID_OCSP_URL	StringValue
TiffBannerCreate	SAPI_ENUM_CONF_ID_TIFF_BANNER	IntegerValue
XAdESTemplate	SAPI_ENUM_CONF_ID_XADES_TEMPLATE	StringValue
TimestampAdditionalBytes	SAPI_ENUM_CONF_ID_SECURED_TS_ADDITIONAL_BYTES	IntegerValue
ExtendedValidation	SAPI_ENUM_CONF_ID_EXTENDED_VALIDATION	IntegerValue
PDFAttribution	SAPI_ENUM_CONF_ID_PDF_ATTRIBUTION	IntegerValue
CertVerify	SAPI_ENUM_CONF_ID_CERT_VERIFY	BinaryValue
GRImageReduce	SAPI_ENUM_CONF_ID_GR_IMG_REDUCE	IntegerValue

ConfIDEnum Value	CoSign Signature Local Equivalent	Parameter Type
MinAdesVerify	SAPI_ENUM_CONF_ID_MIN_ADES_VERIFY	IntegerValue
InfpSpUserName	SAPI_ENUM_CONF_ID_INFP_SP_USER_NAME	StringValue
InfpSpPassword	SAPI_ENUM_CONF_ID_INFP_SP_USER_PWD	StringValue
InfpSpDomain	SAPI_ENUM_CONF_ID_INFP_SP_DOMAIN	StringValue
InfpSpTemplateName	SAPI_ENUM_CONF_ID_INFP_TEMPLATE_NAME	StringValue
AppearanceMask	SAPI_ENUM_CONF_ID_APPEARANCE_MASK	IntegerValue
OldStylePDFApp	SAPI_ENUM_CONF_ID_OLD_STYLE_PDF_APP	IntegerValue
PDFForceDisplaySigner	SAPI_ENUM_CONF_ID_PDF_FORCE_DISPLAY_SIGNER	IntegerValue
GRImgReduceScale	SAPI_ENUM_CONF_ID_GR_IMG_REDUCE_SCALE	IntegerValue
SignatureAdditionalBytes	SAPI_ENUM_CONF_ID_SIGNATURE_ADDITIONAL_BYTES	IntegerValue
XMLAltID	SAPI_ENUM_CONF_ID_XML_ALT_ID	StringValue
PADESEnable	SAPI_ENUM_CONF_ID_PADES_ENABLE	IntegerValue
PDFLocatorOpenPattern	SAPI_ENUM_CONF_ID_PDF_LOCATOR_OPEN_PATTERN	StringValue
PDFLocatorClosePattern	SAPI_ENUM_CONF_ID_PDF_LOCATOR_CLOSE_PATTERN	StringValue
AutoGRImageDisable	SAPI_ENUM_CONF_ID_AUTO_GR_IMAGE_DISABLE	IntegerValue
EmbedCRLChain	SAPI_ENUM_CONF_ID_EMBED_CRL_CHAIN	IntegerValue
MAXCRLSize,	SAPI_ENUM_CONF_ID_MAX_CRL_SIZE	IntegerValue
NextCRLUpdateThreshold	SAPI_ENUM_CONF_ID_NEXT_CRL_UPDATE_THRESHOLD	IntegerValue
CriticalCRLErrors	SAPI_ENUM_CONF_ID_CRITICAL_CRL_ERRORS	IntegerValue
CRLCommTimeout	SAPI_ENUM_CONF_ID_CRL_COMM_TIMEOUT	IntegerValue
OCSPCacheLength	SAPI_ENUM_CONF_ID_OCSP_CACHE_LENGTH	IntegerValue

**Note:** The default value of **OldStylePDFApp** in CoSign Signature Local is 0, while in the CoSign Signature SOAPAPI its default value is 1.

### ExtendedTimeFormatEnum

This type is equivalent to [SAPI\\_ENUM\\_EXTENDED\\_TIME\\_FORMAT](#).

This enumerated type lists the available GMT related time display.

**None** – Not used.

**GMT** – GMT related information will be displayed with the time.

**System** – GMT related information will not be displayed.

### **GraphicImageFormatEnum**

This type is equivalent to [SAPI\\_ENUM\\_GRAPHIC\\_IMAGE\\_FORMAT](#).  
This enumerated type lists all the existing formats of the graphical image.

### **UserCertStatusEnum**

This type is equivalent to [SAPI\\_ENUM\\_CERT\\_STATUS](#).  
This enumerated type lists the available certificate statuses. This status is returned upon a Signature verification request.

**Not Checked** – Certificate was not checked.

**OK** – Certificate status is OK.

**Revoked** – Certificate is revoked.

**Invalid** – Relevant certificate is invalid.

**Warning** – There is a problem with the certificate. For example, the full certificate chain cannot be established.

### **EnrollmentStatusEnum**

This type is equivalent to [SAPI\\_UM\\_ENUM\\_USER\\_ENROLLMENT\\_STATUS](#).  
This enumerated type lists the available enrollment status for a certificate. This status is returned upon a user's query request.

### **UserCertStatusEnum**

This type is equivalent to [SAPI\\_UM\\_ENUM\\_USER\\_CERT\\_STATUS\\_TYPE](#).  
This enumerated type lists the available user certificate status. This status is returned upon a user's query request.

### **UserLoginEnum**

This type is equivalent to [SAPI\\_UM\\_ENUM\\_USER\\_LOGIN\\_STATUS](#).  
This enumerated type lists the available user login status. This status is returned upon a user's query request. This status can be updated.

### **GroupStatusEnum**

This type is equivalent to [SAPI\\_UM\\_ENUM\\_GROUP\\_STATUS\\_TYPE](#).  
This enumerated type lists the available group activity status. This status is returned upon a group's query request. This group status can be updated.

### **PendingRequestStatusEnum**

This type is equivalent to [SAPI\\_UM\\_ENUM\\_PENDING\\_REQUEST\\_STATUS\\_TYPE](#).  
This enumerated type lists the available pending request status of a user when using a WWV CA. This status is returned upon a user's query request.

### **SPML returnDataType**

This type is used in the User Management API operations to specify the content of the response:

**Identifier** – Only the ID is returned.

**Data** – Only data is returned.

**Everything** – Everything is returned.



# CoSign Signature API

---

This section includes the following topics:

[Introduction](#)

[Signing and Verifying](#)

## Introduction

This chapter describes the CoSign Signature network API provided by appliances.

This API is provided starting from CoSign version 7.1 and can be used in addition to the CoSign Signature SOAP interface. The CoSign Signature API is designed to be easier for the developer to use than the CoSign Signature SOAP API.

The API supports signature and validation functions.

Starting from version 8.2, the API will also support user administration operations such as adding a new user, deleting a user, etc.

The CoSign Signature API can be used with all configurations, including Common Criteria EAL4+, and FIPS 140-2 level 3.

Just like the SOAP-based API, the CoSign Signature API can be used with any platform (such as MS Windows, Linux, Solaris), and any development environment (such as Java, .NET, PHP, PERL) that is able to use an HTTPS-based API.

## Technical Information

The service provides the API using HTTPS with TLS (Transport Layer Security) including X.509 Server Authentication Binding.

- The API uses the HTTPS protocol on port 8081.  
To access the API, use: `https://<DNS name of your service>:8081/<API function>`
- The base URL for every request is `https://<DNS name of your service>:8081/sapiws/`  
The specific command name is added after the base URL.  
For example, `https://<DNS name of your appliance>:8081/sapiws/ChangePassword` performs a `SAPICredentialChange` command.
- [User Administration](#) The API functions use either HTTP-GET commands or HTTP-POST commands, as described below.
- Parameters sent to and received from the appliance are encoded using the JavaScript Object Notation (JSON) format.  
When JSON is used, parameters that are not mandatory can be omitted from the JSON structure.
- The HTTP-POST command includes a content-type attribute. If the content type is `application/x-www-form-urlencoded`, the input data will be url decoded prior to processing. If the content type is either `application/json` or `text/plain` there will be no url decoded operation prior to processing the given input parameters.
- Binary information is represented in BASE64 encoding.

## Integrating applications with the Appliance

A special entry in the *hosts* file in the client side can be allocated to *cosign*, which will point to the IP address of a appliance in your network.

You also must install the ARX ROOT certificate, which is available in the CDROM in the *misc* directory. This enables the client to establish an SSL session with the appliance.

The SSL server key and certificate are temporary and should be replaced by the customer when moving to production. For instructions on how to install a new key and certificate for the SSL server, refer to the chapter on managing the appliance in the *DocuSign Signature Appliance Administrator Guide*.

## Signature and Validation API

The CoSign Signature API is very similar to the SAPICrypt functionality, but currently includes only the most commonly-required functions. It focuses on the following:

- Create and sign a signature field in a file
- Sign an existing field in a file
- Verify an existing signature field in a file
- Sign a data buffer
- Verify a signature of a data buffer
- Retrieve user certificates
- Manage graphical signatures
- Validate a user's credentials
- Change a user's password
- Activate a user's account

The following information can be signed and verified:

- Data Buffers
- Hash values of data
- XML data
- PDF files
- Office 2007/2010/2013 files that already contain signature fields
- Word XP/2003 files that already contain file-based signature fields
- TIFF files
- InfoPath 2007/2010/2013 forms that already contain signature fields

## User Administration API

The User Administration API, to be introduced in version 8.2, is very similar to the SAPIUM functionality, but currently includes only the most commonly-required functions. It focuses on the following:

- Adding a new user
- Deleting an existing user

Enabling and Disabling an existing user

Unlocking an existing user

## General Data Structures for CoSign Signature API version 0.1

Every CoSign Signature command is mapped to a SAPICrypt API, which is described in detail in the above SAPICrypt sections.

Whenever a CoSign Signature web service is called, the appliance responds with an answer.

If the answer is successful, the response has the following JSON format:

```
{
  "Success": true ,
  "data":
  {
    OutParam1: data
    OutParam2: data
    ...
    Outparam3: data
  }
}
```

Upon functional failure, the response has the following JSON format:

```
{
  "Success": false,
  "Errdata":
  {
    "Message": error message,
    "Module": the module that returned the error,
    "Code": error code,
    "InnerCode": SAPICrypt error code
  }
}
```

The InnerCode in an Error response presents a SAPICrypt error code (for a full description of the SAPICrypt error codes, refer to [CoSign Signature Local – Signing/Verifying Error Messages](#)).

The basic response returns either a success without replied data, or an error structure.

## General Data Structures for CoSign Signature API version 1.0

Specifying the relevant resources as the subject of the operation. For example, if it is required to delete a certain user then the URL will look as follows:

**Error! Hyperlink reference not valid.**

Where u1 is the user you intend to delete.

Using the HTTP method for the desired operation. For example, in the above case, the HTTP DELETE method is used as part of the HTTP protocol.

Note that the reply to whether the operation was successful or not employs the HTTP response style. For example, a return code of 200 indicates success.

The relevant login user and his/her password pass through the *Authorization* parameter in the HTTP header. The value of the *Authorization* parameter is the base64 encoding of the following parameter *<user ID>:<Password>*

The input content type is: *application/json*

### **SAPIWS\_ENUM\_PASSWORD\_TYPE – Enumerated Values**

The SAPIWS\_ENUM\_PASSWORD\_TYPE defines the format of the login or signature password. This enumerated type is used in the CoSign Signature API.

```
typedef enum{
SAPIWS_PASSWORD_TYPE_NONE           = 0,
SAPIWS_PASSWORD_TYPE_STRING         = 1,
SAPIWS_PASSWORD_TYPE_BINARY         = 2
}SAPIWS_ENUM_PASSWORD_TYPE;
```

## Signing and Verifying – V0.1

This chapter provides a detailed description of the functions available in the Signature interface.

### Create and Sign

This function creates and signs a new signature field.

#### *URL Resource Name*

CreateSign

#### *SAPICrypt Base Functions*

[SAPISignatureFieldSignEx](#)

#### *Input Parameters*

Name	Type	Mandatory	Comments	Default Value
Username	String	Yes		
Domain	String	No		
PasswordType	Int	No	The format of the login password, as defined in the <a href="#">SAPIWS_ENUM_PASSWORD_TYPE</a> enumerated type.	SAPIWS_PASSWORD_TYPE_STRING
Password	String	Yes		
SamlAuth	Boolean	No	If the value is true, the password field should be UTF8 encoded to Base64.	false
FileData	Buffer	Yes	The data to sign	
FileType	string	Yes	The file type can be any one of the following: <ul style="list-style-type: none"> <li>▪ “doc”</li> <li>▪ “docx”</li> <li>▪ “xls”</li> <li>▪ “xlsx”</li> <li>▪ “pdf”</li> <li>▪ “tif”</li> <li>▪ “xml”</li> <li>▪ “inp”</li> </ul>	
SignPasswordType	Int	No	The format of the signature password as defined in the <a href="#">SAPIWS_ENUM_PASSWORD_TYPE</a> enumerated type.	SAPIWS_PASSWORD_TYPE_STRING

Name	Type	Mandatory	Comments	Default Value
SignPassword	String	No	If prompt for sign is enabled, use this password to sign.	
GrSigName	String	No	Graphical signature name.	If not given, selects the first image
UserCertID	String	No	Certificate ID to sign with.	If not given, selects the first certificate
Reason	String	No		
Title	String	No		
UserGMTOffset	Int	No	The GMT offset, in minutes, used for the purpose of the digital signature operation. For example: “-60” is GMT-1.	
SigFieldName	String	No	The name of the field to sign (if it exists). Otherwise, the name of the new signature field to create.	
Width	Int	No	Width of the signature field.	“100”
Height	Int	No	Height of the signature field.	“100”
X	Int	No	Top-Left X coordinate of the signature field.	“100”
Y	Int	No	Top-Left Y coordinate of the signature field.	“100”
Page	Int	No	The number of the page in which to create the field. For example: “1”.	“-1” (last page)
Appearance	Int	No	The appearance mask.	“11”. This means that the following information will be displayed when the signature field is signed: <ul style="list-style-type: none"> <li>▪ Graphical image</li> <li>▪ Signed by</li> <li>▪ Time with difference from GMT</li> </ul>
Type	Int	No	According to the <a href="#">SAPI_ENUM_SIGNATURE_TYPE</a> enumerated type. Use the decimal value of the enumerand.	“1” (that is, digital)

Name	Type	Mandatory	Comments	Default Value
Flags	Int	No	Refer to the flags in the  SAPISignatureFieldCreateSignEx function. Use the decimal value of the Flags and not their hexadecimal value.	“0”
TimeFormat	String	No	The format string to use to form the time string. Refer to <a href="#">TimeFormat</a> .	“h:mm tt”
DateFormat	String	No	The format string used to form the date string. Refer to <a href="#">DateFormat</a> .	“MMM d yyyy”
Labels	Int	No	Label mask. Use the decimal value of the mask and not its hexadecimal value.	0 (which means that labels are not displayed in the visible signature)
HashAlg	String	No	The hash algorithm to use for the signature. The possible values are: <ul style="list-style-type: none"> <li>▪ “Sha1”</li> <li>▪ “Sha256”</li> <li>▪ “Sha384”</li> <li>▪ “Sha512”</li> </ul> <b>Note</b> that when DocuSign Signature Appliance is deployed in Common Criteria mode or in FIPS mode, any attempt to sign using SHA1 is rejected by the Appliance.	“Sha256”
TimeStampUrl	String	No		
TimeStampUser	String	No		
TimeStampPwd	String	No		
RetFileMode	Int	No	0 – The entire signed file is returned. 1 – In the case of PDF files, indicates that only the signature is returned after the signature operation.  By default, the entire file is returned.	0

Name	Type	Mandatory	Comments	Default Value
RetSigFieldMode	Int	No	<p>0 – No signature field information is returned.</p> <p>1 – Only signed field information is returned. This option is available when a new signature is created and signed.</p> <p>2 – All fields information (both signed fields and unsigned fields) is returned.</p>	0
ExtTimeFormat	Int	No	<p>The displayed time format, as specified in the <a href="#">SAPI_ENUM_EXTENDED_TIME_FORMAT</a> enumerated type.</p> <p>Use the decimal value of the enumerand.</p>	0

### ***Output Parameters***

In case of success, the parameters in the following table are returned. In case of error, a regular Error response is returned.

Name	Type	Mandatory	Comments
Signed file data	Buffer	Yes	
NumOfFields	Int	No	The number of returned fields, according to the settings in <a href="#">RetSigFieldMode</a> .
Fields	Array of data structs	No	<p>An array of fields that contains data about each signature field inside the file.</p> <p>For more information, refer to the <a href="#">Get Signature Fields</a> function.</p>

## Sign

This function signs an existing signature field.

### **URL Resource Name**

Sign

### **SAPICrypt Base Functions**

[SAPISignatureFieldSignEx](#)

### **Input Parameters**

Name	Type	Mandatory	Comments	Default Value
Username	String	Yes		
Domain	String	No		
PasswordType	Int	No	The format of the login password, as defined in the SAPIWS_ENUM_PASSWORD_TYPE enumerated type.	SAPIWS_PASSWORD_TYPE_STRING
Password	String	Yes		
SamlAuth	Boolean	No	If the value is true, the password field should be UTF8 Encoded to Base64.	false
FileData	Buffer	Yes	The data to sign.	
FileType	string	Yes	The file type can be any one of the following: <ul style="list-style-type: none"><li>▪ “doc”</li><li>▪ “docx”</li><li>▪ “xls”</li><li>▪ “xlsx”</li><li>▪ “pdf”</li><li>▪ “tif”</li><li>▪ “xml”</li><li>▪ “inp”</li></ul>	
SignPasswordType	Int	No	The format of the signature password, as defined in the SAPIWS_ENUM_PASSWORD_TYPE enumerated type	SAPIWS_PASSWORD_TYPE_STRING
SignPassword	String	No	If prompt for sign is enabled, use this password to sign.	

Name	Type	Mandatory	Comments	Default Value
GrSigName	String	No	Graphical signature name.	If not given, selects the first image
UserCertID	String	No	Certificate ID to sign with.	If not given, selects the first certificate
Reason	String	No		
Title	String	No		
UserGMTOffset	Int	No	The GMT offset used for the purpose of the digital signature operation. For example: “-6”.	
SigFieldName	String	No	The name of the field to sign (if it exists). Otherwise, the name of the new signature field to create.	
HashAlg	String	No	The hash algorithm to use for the signature. The possible values are: <ul style="list-style-type: none"> <li>▪ “Sha1”</li> <li>▪ “Sha256”</li> <li>▪ “Sha384”</li> <li>▪ “Sha512”</li> </ul> <p><b>Note</b> that when DocuSign Signature Appliance is deployed in Common Criteria mode or in FIPS mode, any attempt to sign using SHA1 is rejected by the Appliance.</p>	“Sha256”
TimeStampUrl	String	No		
TimeStampUser	String	No		
TimeStampPwd	String	No		
Flags	Int	No	Refer to the flags in the <a href="#">SAPISignatureFieldSignEx</a> function. Use the decimal value of the Flags and not their hexadecimal value.	0
RetFileMode	Int	No	0 – The entire signed file is returned 1 – In the case of PDF files, indicates that only the signature is returned after the signature operation. By default, the whole file is returned.	0

Name	Type	Mandatory	Comments	Default Value
RetSigFieldMode	Int	No	<p>0 – No signature field information is returned</p> <p>1 – Only signed field information is returned. This option is not available when a new signature is created and signed.</p> <p>2 – All fields information (both signed fields and unsigned fields) is returned</p>	0

### ***Output Parameters***

In case of success, the parameters in the following table are returned. In case of error, a regular Error response is returned.

Name	Type	Mandatory	Comments
Signed file data	Buffer	Yes	
NumOfFields	Int	No	The number of returned fields, according to the settings in <a href="#">RetSigFieldMode</a> .
Fields	Array of data structs	No	An array of fields that contains data about each signature field inside the file. For more information, refer to the <a href="#">Get Signature Fields</a> function.

## Get Signature Fields

Gets a list of all signature fields in the file as well as their details.

### *URL Resource Name*

GetSignatureFields

### *SAPICrypt Base Functions*

[SAPISignatureFieldEnumInitEx](#), [SAPISignatureFieldEnumCont](#), [SAPISignatureFieldInfoGet](#)

### *Input Parameters*

Name	Type	Mandatory	Comments	Default Value
FieldName	String	No	If given, returns the details of this field only.  Otherwise, returns the details of all fields.	
FieldState	String	No	The possible values are: <ul style="list-style-type: none"><li>▪ “Signed”</li><li>▪ “unsigned”</li></ul> If no value is provided, all signature fields are returned.	
FileData	Buffer	Yes	The file’s data in Base64 format.	
FileType	String	Yes	File type can be any one of the following: <ul style="list-style-type: none"><li>▪ “doc”</li><li>▪ “docx”</li><li>▪ “xls”</li><li>▪ “xlsx”</li><li>▪ “pdf”</li><li>▪ “tif”</li><li>▪ “xml”</li><li>▪ “inp”</li></ul>	

### *Output Parameters*

In case of success, the parameters in the following table are returned. In case of error, a regular Error response is returned.

Name	Type	Mandatory	Comments
NumOfFields	Int	Yes	

Name	Type	Mandatory	Comments
Fields	Array of data structs	Yes	An array of fields that contains the following data for each signature field inside the file.
Name	String	Yes	Field name
Width	Int	Yes	
Height	Int	Yes	
X	Int	Yes	
Y	Int	Yes	
Page	Int	Yes	
Appearance	Int	Yes	The appearance mask of the field.
Type	Int	Yes	Signature type: Digital or electronic. Refer to <a href="#">SAPI_ENUM_SIGNATURE_TYPE</a> for a description of the possible values.
Flags	Int	Yes	
TimeFormat	String	Yes	
DateFormat	String	Yes	
ExtTimeFormat	Int	Yes	Refer to <a href="#">SAPI_ENUM_EXTENDED_TIME_FORMAT</a> for a description of the possible values.
LabelsMask	Int	Yes	
IsSigned	Boolean	Yes	Whether the field is signed
SignedFieldProperties	Struct	No	A struct that contains the signed field data. It contains the following data.
IsValid	Boolean	No	For signed field
Reason	String	No	For signed field
CertificateStatus	Int	No	For signed field Refer to <a href="#">SAPI_ENUM_CERT_STATUS</a> for a description of the possible values.
Subject	String	No	For signed field
Email	String	No	For signed field
SignatureTime	String	No	For signed field

<b>Name</b>	<b>Type</b>	<b>Mandatory</b>	<b>Comments</b>
IssuedBy	String	No	For signed field
CertExpirationDate	String	No	For signed field

## Sign Buffer

This function signs a buffer and returns the signature.

### **URL Resource Name**

SignBuffer

### **SAPICrypt Base Functions**

[SAPIBufferSignEx](#)

### **Input Parameters**

Name	Type	Mandatory	Comments	Default Value
Username	String	Yes		
Domain	String	No		
PasswordType	Int	No	The format of the login password, as defined in the SAPIWS_ENUM_PASSWORD_TYPE enumerated type	SAPIWS_PASSWORD_TYPE_STRING
Password	String	Yes		
SamlAuth	Boolean	No	If the value is true, the password field should be UTF8 Encoded to Base64	false
BufferToSign	Buffer	Yes		
BufferHash	Boolean	No	Whether the buffer to sign is a hash value or data.	“false” – the buffer is data
SignPasswordType	Int	No	The format of the signature password, as defined in the SAPIWS_ENUM_PASSWORD_TYPE enumerated type	SAPIWS_PASSWORD_TYPE_STRING
SignPassword	String	No	If prompt for sign is enabled, use this password to sign.	
UserCertID	String	No	Certificate ID with which to sign.	If not given, selects the first certificate

Name	Type	Mandatory	Comments	Default Value
HashAlg	String	No	<p>The hash algorithm to use for the signature. The options are:</p> <ul style="list-style-type: none"> <li>▪ “Sha1”</li> <li>▪ “Sha256”</li> <li>▪ “Sha384”</li> <li>▪ “Sha512”</li> </ul> <p><b>Note</b> that when DocuSign Signature Appliance is deployed in Common Criteria mode or in FIPS mode, any attempt to sign using SHA1 is rejected by the Appliance.</p>	“Sha256”
TimeStampUrl	String	No		
TimeStampUser	String	No		
TimeStampPwd	String	No		
Flags	Int	No	<p>Refer to the flags in the <a href="#">SAPIBufferSignEx</a> function.</p> <p>Use the decimal value of the Flags and not their hexadecimal value.</p>	“0”

### **Output Parameters**

In case of success, the parameters in the following table are returned. In case of error, a regular Error response is returned.

Name	Type	Mandatory	Comments
Signature	Buffer	Yes	The signature data.

## Verify Buffer

This function verifies the buffer and returns a list of all signatures.

### **URL Resource Name**

VerifyBuffer

### **SAPICrypt Base Functions**

[SAPIBufferVerifySignature](#)

### **Input Parameters**

Name	Type	Mandatory	Comments	Default Value
BufferHash	Boolean	No	Whether the buffer that was signed is the hash value or the clear data.	“false” – the buffer is data
BufferToSign	Buffer	Yes	The buffer that was signed.	
Signature	Buffer	Yes	The signature value.	
Flags	Int	No	Refer to the flags in the <a href="#">SAPIBufferVerifySignature</a> function. Use the decimal value of the flags and not their hexadecimal value.	“0”
Certificate	Buffer	No	Relevant only in the case of PKCS#1 validation	

### **Output Parameters**

In case of success, the parameters in the following table are returned. In case of error, a regular Error response is returned.

Name	Type	Mandatory	Comments
IsValid	Boolean	Yes	
CertificateStatus	Int	Yes	
Subject	String	Yes	
Email	String	Yes	
SignatureTime	String	Yes	
IssuedBy	String	Yes	
CertExpirationDate	String	Yes	

## Get User Certificates

This function returns all user certificates.

### URL Resource Name

UserCertificatesGet

### SAPICrypt Base Functions

[SAPICertificatesEnumInit](#), [SAPICertificatesEnumCont](#)

### Input Parameters

Name	Type	Mandatory	Comments	Default Value
Username	String	Yes		
Domain	String	No		
PasswordType	Int	No	The format of the login password, as defined in the SAPIWS_ENUM_PASSWORD_TYPE enumerated type	SAPIWS_PASSWORD_TYPE_STRING
Password	String	Yes		
SamlAuth	Boolean	No	If the value is true , the password field should be UTF8 Encoded to Base64	false

### Output Parameters

In case of success, the parameters in the following table are returned. In case of error, a regular Error response is returned.

Name	Type	Mandatory	Comments
NumOfCerts	Int	Yes	
Certificates	Array of structs	Yes	An array of Certificates that contains the data listed below
Subject	String	Yes	
Issuer	String	Yes	
Email	String	Yes	
NotBefore	String	Yes	The first day of the certificate's validity period
NotAfter	String	Yes	The last day of the certificate's validity period
SerialID	String	Yes	

## Get Graphical Signatures

This function gets a list of all the user's graphical signature images.

### URL Resource Name

GrSigImgGet

### SAPICrypt Base Function

[SAPIGraphicSigImageEnumInit](#), [SAPIGraphicSigImageEnumCont](#), [SAPIGraphicSigImageInfoGet](#)

### Input Parameters

Name	Type	Mandatory	Comments	Default Value
Username	String	Yes		
Domain	String	No		
PasswordType	Int	No	The format of the login password, as defined in the SAPIWS_ENUM_PASSWORD_TYPE enumerated type	SAPIWS_PASSWORD_TYPE_STRING
Password	String	Yes		
SamlAuth	Boolean	No	If the value is true, the password field should be UTF8 Encoded to Base64	false
GrSigName	String	No	If given, returns the details of this signature image only; otherwise, returns the details of all the user's signature images.	

### Output Parameters

In case of success, the parameters in the following table are returned. In case of error, a regular Error response is returned.

Name	Type	Mandatory	Comments
NumOfSigs	Int	Yes	
GrSigImages	Array of data structs	Yes	An array of graphical signatures that contains the data listed below.
Name	String	Yes	
GrImageData	Buffer	Yes	
DataFormat	Int	Yes	The graphical image information, presented in the <a href="#">SAPI_GR_IMG_INFO</a> structure.

Name	Type	Mandatory	Comments
Width	Int	Yes	
Height	Int	Yes	

## Create Graphical Signature

This function creates a new graphical signature for a user.

### URL Resource Name

GrSigImgCreate

### SAPICrypt Base Functions

[SAPIGraphicSigImageInfoCreate](#)

### Input Parameters

Name	Type	Mandatory	Comments	Default Value
Username	String	Yes		
Domain	String	No		
PasswordType	Int	No	The format of the login password, as defined in the SAPIWS_ENUM_PASSWORD_TYPE enumerated type	SAPIWS_PASSWORD_TYPE_STRING
Password	String	Yes		
SamlAuth	Boolean	No	If the value is true, the password field should be UTF8 Encoded to Base64	false
GrSigName	String	Yes		
GrImageData	Buffer	Yes		
ReducePercentage	Int	No	The value in percentage by which the graphical signature image is decreased in order to fit the image size limitation.	“0”
ReduceIterations	Int	No	The maximum number of image size reductions allowed.	“0”

### Output Parameters

In case of success, the parameters in the following table are returned. In case of error, a regular Error response is returned.

Name	Type	Mandatory	Comments
NewGrSigImg	Data Struct	Yes	A structure that contains the new graphical signature data as listed below.

Name	String	Yes	
GrImageData	Buffer	Yes	
DataFormat	Int	Yes	
Width	Int	Yes	
Height	Int	Yes	

## Delete Graphical Signature

This function deletes a specific graphical signature.

### *URL Resource Name*

GrSigImgDelete

### *SAPICrypt Base Functions*

[SAPIGraphicSigImageInfoDelete](#)

### *Input Parameters*

Name	Type	Mandatory	Comments	Default Value
Username	String	Yes		
Domain	String	No		
PasswordType	Int	No	The format of the login password, as defined in the SAPIWS_ENUM_PASSWORD_TYPE enumerated type	SAPIWS_PASSWORD_TYPE_STRING
Password	String	Yes		
SamlAuth	Boolean	No	If the value is true, the password field should be UTF8 Encoded to Base64	false
GrSigName	String	Yes	The name of the graphical signature to delete.	

### *Output Parameters*

The default Success response with no data, or an Error response.

## Validate Credentials

This function verifies a given user's credentials.

### **URL Resource Name**

ValidateCredentials

### **SAPICrypt Base Functions**

[SAPILogonEx](#)

### **Input Parameters**

Name	Type	Mandatory	Comments	Default Value
Username	String	Yes		
Domain	String	No		
PasswordType	Integer	No	Possible values are as defined in the SAPIWS_ENUM_PASSWORD_TYPE enumerated type	SAPIWS_PASSWORD_TYPE_STRING
Password	String	Yes		
SamlAuth	Boolean	No	If the value is true, the password field should be UTF8 Encoded to Base64	false

### **Output Parameters**

The default Success response with no data, or an Error response.

## Change Password

This function changes the user password.

### ***URL Resource Name***

ChangePassword

### ***SAPICrypt Base Functions***

[SAPICredentialChange](#)

### ***Input Parameters***

Name	Type	Mandatory	Comments	Default Value
Username	String	Yes		
Password	String	Yes	The user's old password	
PasswordType	Integer	No	Possible values are as defined in the SAPIWS_ENUM_PASSWORD_TYPE enumerated type	SAPIWS_PASSWORD_TYPE_STRING
NewPassword	String	Yes	The user's new password.	
NewPasswordType	Integer	No	Possible values are as defined in the SAPIWS_ENUM_PASSWORD_TYPE enumerated type	SAPIWS_PASSWORD_TYPE_STRING

### ***Output Parameters***

The default Success response with no data, or an Error response.

## User Activate

This function performs user activation.

### URL Resource Name

UserActivation

### SAPICrypt Base Functions

[SAPIUserActivate](#)

#### Input Parameters

Name	Type	Mandatory	Comments	Default Value
Username	String	Yes		
Password	String	Yes		
PasswordType	Integer	No	Possible values are as defined in SAPIWS_ENUM_PASSWORD_TYPE	SAPIWS_PASSWORD_TYPE_STRING
NewPassword	String	Yes	The new password to set after activation.	
ExtCreds	String	No	The extended password (OTP).	
NewPasswordType	Integer	No	Possible values are as defined in SAPIWS_ENUM_PASSWORD_TYPE	SAPIWS_PASSWORD_TYPE_STRING

#### Output Parameters

The default Success response with no data, or an Error response.

## Get Server Info

Returns information about the connecting server.

### *URL Resource Name*

ServerInfoGet

### *SAPICrypt Base Functions*

[SAPITimeGet](#) is partially supported.

### *Input Parameters*

Name	Type	Mandatory	Comments	Default Value
Flags	Int	No	Reserved for future use	

### *Output Parameters*

In case of success, the parameters in the following table are returned. In case of error, a regular Error response is returned.

Name	Type	Mandatory	Comments
RestApiVersion	Data structure	Yes	SAPI RESTful Web Services API minor version
MinorVersion	Int	Yes	SAPI RESTful Web Services API major version
Major Version	Int	Yes	
FirmwareVersion	Data structure	Yes	
▪ MinorVersion	Int	Yes	
▪ MajorVersion	Int	Yes	
HardwareVersion	Data structure	Yes	
▪ MinorVersion	Int	Yes	
▪ Major Version	Int	Yes	
SerialNumber	String	Yes	
ServerTime	String	Yes	Server time in the format YYYY-MM-DDTHH:MM:SSZ
InstallStatus	Int	Yes	
ServerKind	Int	Yes	As defined in <a href="#">SAPI_ENUM_SERVER_KIND</a>
DirectoryKind	Int	Yes	As defined in <a href="#">SAPI_ENUM_DIRECTORY_KIND</a>
SubDirectoryKind	Int	Yes	

Name	Type	Mandatory	Comments
AuthMode	Int	Yes	As defined in <a href="#">SAPI_ENUM_AUTH_MODE</a>
AuthMode2	Int	Yes	As defined in <a href="#">SAPI_ENUM_AUTH_MODE</a>
ClusterId	Int	Yes	

## External CA enrollment – V0.1

This chapter provides a detailed description of the functions available in the Signature interface for the enrollment of a new signature key and certificate.

### Generate key pair

This function generates a signature key for the logged in user.

### URL Resource Name

GenerateKeyPair

### SAPICrypt Base Functions

[SAPIKeyPairGenerate](#)

### Input Parameters

Name	Type	Mandatory	Comments	Default Value
Username	String	Yes		
Domain	String	No		
PasswordType	Int	No	The format of the login password, as defined in the SAPIWS_ENUM_PASSWORD_TYPE enumerated type	SAPIWS_PASSWORD_TYPE_STRING
Password	String	Yes		
Flags	Int	No	Key size: <ul style="list-style-type: none"><li>0 – 2048 key size</li><li>1 – 4096 key size</li></ul>	0

### Output Parameters

In case of success, the parameters in the following table are returned. In case of error, a regular Error response is returned.

Name	Type	Mandatory	Comments
PublicKey	String	Yes	The user's public key in base64 encoding
ContainerName	String	Yes	The signature key identifier

## Import Certificate

This function generates a signature key for the logged in user.

### ***URL Resource Name***

ImportCertificate

### ***SAPICrypt Base Functions***

[SAPIImportCertificate](#)

### ***Input Parameters***

Name	Type	Mandatory	Comments	Default Value
Username	String	Yes		
Domain	String	No		
PasswordType	Int	No	The format of the login password, as defined in the SAPIWS_ENUM_PASSWORD_TYPE enumerated type	SAPIWS_PASSWORD_TYPE_STRING
Password	String	Yes		
Certificate	String	Yes	The user's certificate in Base64 format	

### ***Output Parameters***

In case of success, the parameters in the following table are returned. In case of error, a regular Error response is returned.

## **User Administration – V1.0**

This chapter provides a detailed description of the functions available in the user administration interface. This API will be available from version 8.2.

### **UserAdd**

**UserDisable or UserEnable**

## UserResetPasswordCounter



# Web Agent API

---

This section includes the following topics:

[Introduction](#)

[Interface between your Web Application and Web Agent](#)

[XML and Redirection URL Samples](#)

## Introduction

The Web App is a UI web based ASPX.NET product that handles Digital Signing and Signature Verification operations. The Web App provides the Web Agent API. The Web Agent API is the fastest and easiest way to add open, standard digital signatures to your application, using any language and any platform.

The API includes a full User Experience (UX) layer. The UX enables the user to preview the document to be signed, dynamically locate and size his signature, and then sign the document. You can also enable the UX to provide the user with management features: update the user's graphical signatures, signing reasons and more.

The Web Agent handles all user experience aspects of previewing and digitally signing documents. Your web app controls the Web Agent by sending requests and receiving responses via HTTPS. The requests and responses are sent in XML data format.

The Web Agent API enables web applications to easily integrate with DocuSign Signature Appliance in order to provide a web based signing solution as well as a signature verification service.

The Web App is designed to be used either directly by a person, or integrated with your web application. When it is integrated with your web application, we refer to the product as the Web Agent, because it is acting as an agent of your web app. Your web app controls the Web Agent through the Web Agent API.

The PKCS#7 or PKCS#1 signature is then validated using the signature validation function.

**Important:** The Web Agent API is not enabled by default. To use the API, you must enable it from the Web App's web config file. For information, see [Setting Basic Integration Settings in the Web App Configuration File](#).

## Typical Web App Implementations

The applications that can typically benefit from integrating with the Web App are document management applications, business process applications and workflow applications that are based on a web application. In these applications, the user usually has a web based session with the application, which requires a digital signature operation. One option for implementing the digital signature operation is for the application to use the various previously-mentioned Signature APIs (CoSign Signature Local, CoSign Signature SOAP, CoSign Signature). However, an easier method of integrating digital signature operations is to redirect the user as part of the web session to the Web App for a digital signature operation and when the user is done, send him/her back to the original web application to continue his/her work there with the signed document.

In this model, the Web App handles all of the UI aspects of the signing ceremony, including visual dragging and re-sizing of the signature location.

## Required Components

In order to implement an integration based on the Web App, the following components are required:

A web application that communicates with the Web App.

A Web App installed on a Windows 2008R2 server or a higher version.

A set of user accounts on a appliance. The appliance is the engine that performs the actual digital signature operation.

**Note:** The Web App can only work with a Appliance version 6.2 or above.

## Overview of the Signing Process

After integrating your web application with Web App, the signing process proceeds as follows:

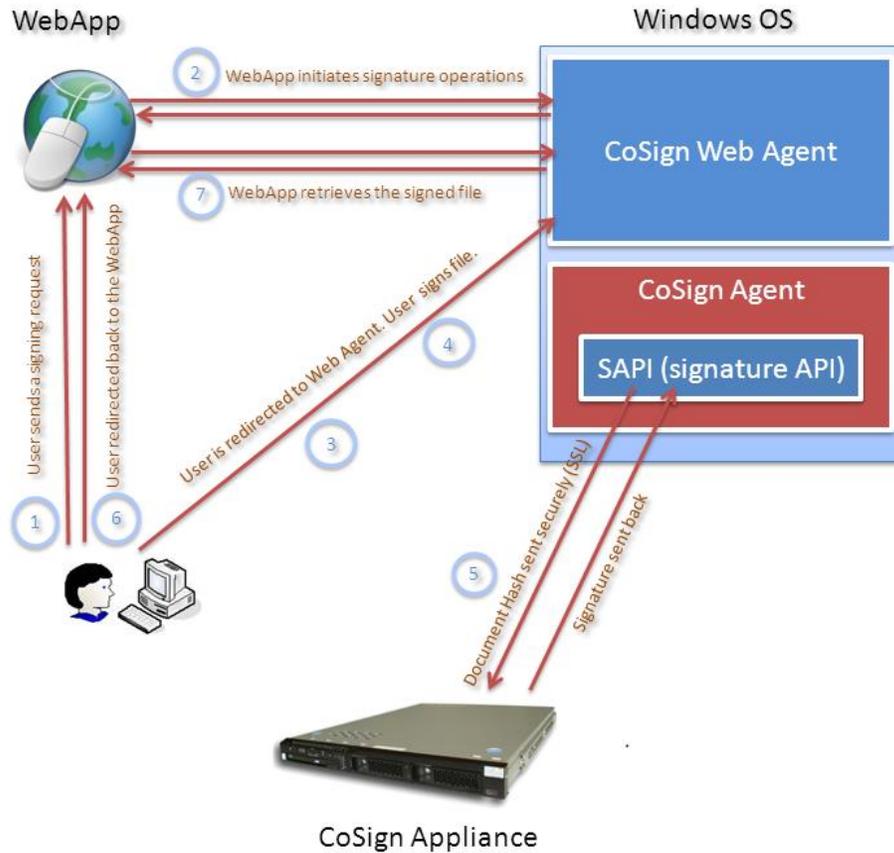
1. A user initiates a signing request to your web application.
2. Your web application sends Web App a file embedded inside an XML in an HTTP-POST request. The XML request includes various parameters (described in the following sections) that influence the signature process.  
The Web App assigns a SessionID which is returned to your web application.
3. Your web application redirects the user's browser to the Web App.
4. The user logs on to the Web App and views the uploaded file. The user can browse the file, sign empty signature fields, and create new signatures. Note that user login to Web App can be done as part of the user session with the web application; in this case, the XML HTTP-POST command includes the user login information.

The signing process takes place between the browser and the Web App according to the HTTP request parameters. Note:

- ◆ The document types supported for signing are PDF, DOC, DOCX, XLS and XLSX. Any non-PDF file is first converted to PDF and then the PDF file is signed. The signed file returned to your web application is always a PDF file. Only DOCX and XLSX files that contain signatures or signature fields are not converted to PDF and are signed using the native Microsoft Office signature provider.
  - ◆ As part of the user's session with the Web App, a new graphical signature can be uploaded to DocuSign Signature Appliance and used.
5. The Web App communicates with the appliance for producing a digital signature.

The user selects the Done option or the Reject option. The Reject option is only relevant for a Signature Workflow scenario.

6. The user is redirected back to the web application.
7. Upon success, the web application retrieves the signed file from the Web App.



**Signing Process Flow Using Web Agent API**

## Overview of the Verification Process

After integrating your web application with Web App, the verification process proceeds as follows:

1. The web application sends Web App a file embedded inside an XML in an HTTP-POST request, and requests signature validation. The XML request includes the file information (as described in the following sections).  
This operation is supported for signed PDF, DOCX and XLSX files.
2. Web App performs digital signature verification as follows:  
  
All digital signatures in the document are validated.
3. An XML structure is returned, containing an individual status per each signature field as well as a general validity status.

## Setting Basic Integration Settings in the Web App Configuration File

The **APISettings** section of the Web App configuration file (web.config) defines the basic settings for integrating with web applications. These include:

Key	Description	Default	Mandatory
EnableAPIRequests	Whether to enable API request handling by the Web App application.	False	Yes
APIRequestTimeOut	The API request session time out. This is the amount of time in minutes the Web App keeps the signed file to be retrieved by the web application after the user finished signing the document.	30 minutes	No

To enable integrating your web application with Web App, you must set **EnableAPIRequests** to **True**.

For more information about the web.config file, refer to the *Web App User Guide*.

## Interface between your Web Application and Web Agent

This chapter describes in detail the Web Agent protocol used between your web application and Web Agent. The protocol defines the parameters and methods to use when performing the following actions:

Upload File and Initiate the Signing Ceremony (Step 2 in [Signing Process Flow](#)). This is comprised of two actions:

- ◆ A request sent by your web application to sign a specific file.
- ◆ A response returned by Web Agent which includes the session ID of the newly created session.

Redirect user to Web Agent (Step 3 in [Signing Process Flow](#)). Starting from this point, the user interacts with the Web Agent for the purpose of signing the uploaded file.

Redirect user back to your web application when the user has completed the signing process (Step 6 in [Signing Process Flow](#)).

Download Signed file (Step 7 in [Signing Process Flow](#)). This is comprised of two actions:

- ◆ A request sent by your web application to receive the signed file.
- ◆ A response returned by Web Agent which includes the signed file.

Upload File and Verify Signatures. This is comprised of two actions:

- ◆ A request sent by your web application to verify signatures in a specific file.
- ◆ A response returned by Web Agent which includes validity information about the signatures in the file.

### The Networking Infrastructure

All requests and responses use the HTTPS protocol. Depending on the operation, HTTP-GET, HTTP-POST or HTTP-Redirect commands are used. All Commands are HTTP protocol commands, secured with the SSL/TLS protocol.

The entire user experience is web based:

The user starts his/her session with your web application. When a digital signature is needed, the user is redirected to the Web Agent using the HTTP-Redirect command.

When signing is completed, the user is redirected back to your web application using an HTTP-Redirect command, and continues his/her user session.

## Uploading a File and Initiating Signing

The following sections describe the format of the request and response when uploading a file and initiating signing.

### ***Request Format***

To initiate the file upload and signing process, an XML request must be sent from your web application to: `https://<host name>/Sign/UploadFileToSign`

The HTTP header of the request must specify the POST method, and it must include a Content-Type parameter with a value `application/x-www-form-urlencoded`. The actual data of the HTTP-POST command should include the parameter `inputXML`, and the content should be an XML structure. The XML structure must contain all relevant data regarding the desired behavior of the Web Agent, as well as the file to be signed, Base64 encoded. For a full description of the parameters in the XML structure, refer to [Request Parameters](#).

For an example of such an XML structure, refer to [Sample XML Request](#).

### ***Authentication Options***

To digitally sign a document with DocuSign Signature Appliance, a user must authenticate to the signing system by presenting signing credentials. Each installation of the Web Agent can be configured to use a appliance. DocuSign Signature Appliance supports various modes of authentication (as described in the *DocuSign Signature Appliance Administrator Guide*). The role of the Web Agent in terms of authentication is to provide DocuSign Signature Appliance with the information necessary to authenticate a user. Currently, Web Agent supports username/password authentication mode. Within this mode, Web Agent provides the following options for receiving user credentials:

- The user enters his/her username and password in the Web Agent's Login page when redirected to the Web Agent.

- Your web application provides the username in the request XML. The login page will display the username as "read only" and the user will only need to enter his/her password (in this case, a user will not be able to enter a different username).

- Your web application provides both the username and password in the request XML. If the username and password are valid, the user is redirected to the Signing Ceremony page and the Login Page is skipped. If the credentials are not valid, the operation is aborted and an error is returned to the web application as part of response returned by the Web Agent.

## Request Parameters

The request parameters are sorted by logical groups. Some of these groups can have multiple instances in a single request. For example, a request can specify a number of Sign Reasons for the user to choose from. For an example of a request, refer to [Sample XML Request](#).

Group	Name	Type	Description	Default if not Specified	Notes
<b>Auth</b>	username	String	Username	Request user for input	
	password	String	Password of a user	Request user for input	
	keepUserLogin	Boolean	Specifies whether the user remains logged-in after signing the document and can sign the next document without presenting his/her credentials again.  Possible values: <b>true</b> , <b>false</b> .	False	
<b>Document</b>  This group is required	content	String	A Base64 encoded file to be signed		Mandatory
	contentType	String	The type of document to be signed. Supported types: <b>PDF</b> , <b>Doc</b> , <b>Docx</b> , <b>Xls</b> , <b>Xlsx</b> .		Mandatory
	fileID	String	A unique identifier for the file. This identifier will be returned after a file was signed in order for the calling application to identify it.		
	fileName	String	The document file name. This will be displayed to the user in the Signing Ceremony page.  Note that the value of this parameter is used for display purposes only and does not determine the document type. The document type is specified using the <b>contentType</b> parameter.		

Group	Name	Type	Description	Default if not Specified	Notes
Layout	layoutMask	Int	Determines the sign ceremony menu displayed to the user. A mask of the following options can be passed: None = 0 Home/Select Document = 1 Education center = 2 Settings and change password = 8 Graphical Signatures = 16 Logoff option = 32 App Title = 128	0	
UserGMTOffset	GMTOffset	Int	Determines the user GMT offset. Used in case the user's GMT offset is different from the server system settings.  The value is in minutes.		
SigField	fieldNameToSign	String	Specifies a name of a signature field in the document or a signature profile in a request.  If <b>enforceFieldToSign</b> is true, this is the name of the signature field that must be signed; otherwise, the user does not have to sign this field.  This parameter is not allowed in a Point of Sale signing process.		
	enforceFieldToSign	Boolean	Specifies whether the user must sign the field specified in the <b>fieldNameToSign</b> parameter.  If true, the user cannot select another existing field or create a new field to sign, nor can the user change the given signature's profile settings.  Possible values: <b>true, false</b> .  This parameter is not allowed in a Point of Sale signing process.	False	

Group	Name	Type	Description	Default if not Specified	Notes
<b>SigProfiles/SigProfile</b> (The <b>SigProfiles</b> group contains a single <b>SigProfile</b> group)	x	Int	X coordinate of a signature field to be created. X should be a positive integer.	Value required if group exists	This entire group of parameters is not allowed when signing Office files.
	y	Int	Y coordinate of a signature field to be created. X should be a positive integer.	Value required if group exists	
	width	Int	Specifies the width of the signature field to be created. X should be a positive integer.	Value required if group exists	
	height	Int	Specifies the height of the signature field to be created. X should be a positive integer.	Value required if group exists	
	pageNumber	Int	Specifies the page number on the document	Value required if group exists	
	fieldName	String	The name of the field that will be created. <b>fieldName</b> is limited to 32 characters and supports only English, digits, +, -, and =.	Value required if group exists	
	isNewFieldDefault	Boolean	If true, the signature profile settings will be used as default settings for the ad-hoc (newly created) fields. This means the user will be able to move, resize and change the appearance settings of the field. Note that in order to enable the user to create and sign new fields the <a href="#">allowAdHoc</a> parameter must be set to <b>true</b> .	False	
	dateFormat	String	The format string used to form the date string. Refer to <a href="#">DateFormat</a> .	MMM d yyyy	
	timeformat	String	The format string to use to form the time string. Refer to <a href="#">TimeFormat</a> .	h:mm tt	

Group	Name	Type	Description	Default if not Specified	Notes
	isDisplayGMT	Boolean	Specifies whether to display GMT-related information in addition to the displayed date and time.  Possible values: <b>true, false.</b>	False	
	signer	Boolean	Specifies whether to display the Signed By value (Printed Name).  Possible values: <b>true, false.</b>	True	
	date	Boolean	Specifies whether the date should be visible.  Possible values: <b>true, false.</b>	True	
	graphicalImage	Boolean	Specifies whether to display a graphical image.  Possible values: <b>true, false.</b>	True	
	showTitle	Boolean	Specifies whether to show the title in the signature field.  Possible values: <b>true, false.</b>	False	
	title	String	A title that will appear in the signature field if <b>showTitle</b> is True. Title length is limited to 64 characters. Note that the user will be able to change this during the signing ceremony.	Request user for input	
<b>Sign Reasons</b> (One or more occurrences within the XML)	signReason	String	Determines the reasons in the Reasons combo box.  Each <b>signReason</b> occurrence is a separate reason. The length of <b>signReason</b> is limited to 128 characters.  Note that in order to enable the user to enter his/her own sign reason instead of selecting from the combo-box, the <b>allowUserReason</b> parameter must be set to <b>true</b> .	Request user for input	Multiple instances

Group	Name	Type	Description	Default if not Specified	Notes
<b>Reject Reasons</b> (One or more occurrences within the XML)	rejectReason	String	<p>Determines the reasons in the Reject combo box.</p> <p>Each <b>rejectReason</b> occurrence is a separate reason. The length of <b>rejectReason</b> is limited to 128 characters.</p> <p>Note that in order to enable the user to enter his/her own rejection reason instead of selecting from the combo-box, the <b>allowUserReason</b> parameter must be set to <b>true</b>.</p>	Request user for input	Multiple instances
<b>Logic</b>	isRejectButton	Boolean	<p>This parameter affects the UI. When true, it adds a Reject button so that the user can either sign the document or click Reject.</p> <p>Possible values: <b>true, false</b>.</p> <p>This parameter is not allowed in a Point of Sale signing process.</p>	False	
	allowAdHoc	Boolean	<p>Allow users to dynamically create a signature field. If the signature profile settings are given and <a href="#">isNewFieldDefault</a> is true, those settings will determine the new field's default settings.</p> <p>Possible values: <b>true, false</b>.</p> <p>This parameter is disregarded when signing Office files.</p>	False	
	IsWorkflowMode	Boolean	<p>If this parameter is set to True, then the user must either Sign or Reject the signature as part of a workflow.</p> <p>This parameter is not allowed in a Point of Sale signing process.</p>	False	
	enforceReason	Boolean	<p>Specifies whether a user must enter a reason.</p> <p>Possible values: <b>true, false</b>.</p>	True	

Group	Name	Type	Description	Default if not Specified	Notes
	allowUserReason	Boolean	Specifies whether the user can enter his/her own reason text, rather than selecting a reason from the predefined reasons list.  Possible values: <b>true, false.</b>	True	
	verify	Boolean	Specifies whether to verify all existing signatures in the document in addition to the signing process.  If you specify true, a <b>Verify</b> XML node is added to the sign response XML. The structure of that node is identical to the structure received when sending a separate Verify Signatures request (refer to <a href="#">Response Format</a> ).  Note that specifying <b>true</b> may impact performance.	False	
	posMode	Boolean	Specifies whether to enable a Point of Sale signing process. This parameter affects the UI behavior and user signing options.  In Point of Sale mode, a customer can sign electronically using his handwritten signature.  Point of Sale mode is relevant only for PDF files and is not allowed when signing Office files.  Possible values: <b>true, false.</b>	False	
<b>Url</b>	finishURL	String	The URL to which the Web Agent redirects the browser after the user finishes signing.		Mandatory

Group	Name	Type	Description	Default if not Specified	Notes
This group is required	redirectIFrame	Boolean	Relevant when the Web Agent is running on an iframe. If <b>true</b> , <i>only</i> the iframe will redirect to the page specified in <a href="#">finishURL</a> . Otherwise, the whole page will redirect to the <a href="#">finishURL</a> .	False	

### **Response Format**

The response will be based on the following format:

*Content Type = application/xml; charset=utf-8*

### **Response Parameters**

The XML response will include the following fields:

Group	Name	Type	Description
<b>Error</b>	returnCode	Int	0 – specifies success. This means that the document and signature data were successfully received, and the user can now be redirected to Web Agent.  For a list of all possible error codes, refer to <a href="#">Web Agent API Error Codes</a> .
	errorMessage	String	The error message in case of an error.
<b>Session</b>	sessionId	String	A unique session identifier. Used to maintain a session between the Web Agent and your web application.
	docId	String	The ID of the document that was passed to the <b>Sign/UploadFileToSign</b> service. Refer to the <a href="#">fileID</a> parameter in <a href="#">Request Parameters</a> .

## Redirecting a User to Web Agent

The user should be directed to the Web Agent using the following URL:

```
https://<host name>/Sign/SignCeremony?sessionId==1212122
```

where *sessionId* is a value returned from the Web Agent in the Initiate Signing Response (see [Response Format](#) described above).

The user will be able to view the uploaded document, browse different pages and then sign the relevant signature field(s) or create and sign a new signature. The Signing Ceremony will proceed according to the parameters sent in [Request Parameters](#).

## Redirecting a User Back to your Web Application

After signing operations are completed, the Web Agent redirects the web browser to the page specified in [finishURL](#). The redirection includes the following HTTP-GET parameters:

**sessionId** – request identifier

**docId** – document identifier

**returnCode** – sign operation outcome:

- ◆ 0 – success
- ◆ -1 – general error
- ◆ -2 – cancelled
- ◆ -3 – rejected
- ◆ other number – signing error code (refer to [Web Agent API Error Codes](#)).

**errorMessage** – error message or reject reason (refer to [Web Agent API Error Codes](#)).

Note that an error message is only included if the **returnCode** is not 0.

For example:

```
https://webapp.org.com/finishURL.aspx?sessionId=12321_3213213&docId=12321&returnCode=0
```

Your web application can now continue and download the signed file from the Web Agent as described in [Downloading a Signed File](#).

## Downloading a Signed File

Upon successful document signing, your web Application can request the Web Agent to return an XML response that includes the signed document.

### Request Format

If the file was signed successfully, your web application can pull the signed file as part of a returned signing response XML. The web application calls the **Sign/DownloadSignedFileG** service with the session ID parameter using an HTTP-GET method. For example:

```
https://cosign-web-app.org.com/Sign/DownloadSignedFileG?sessionId=12321_3213213
```

### Response Format

The following table describes the various parameters used by the response protocol. The response is sent from the Web Agent to your web app.

For an example of such an XML structure, refer to [Sample Response](#).

Group	Name	Type	Description
<b>Result</b>	operationStatus	Int	1 – signed.
<b>Error</b>	returnCode	Int	0 – specifies success. For a list of all possible error codes, refer to <a href="#">Web Agent API Error Codes</a> .
<b>Session</b>	sessionId	String	The session identifier of the request. .
	docId	String	The ID of the document that was passed to the <b>Sign/UploadFileToSign</b> service.
<b>SigDetails</b> (Details of the latest signature)	fieldName	String	The name of the latest field that was signed. To retrieve information about all the fields that were signed, send a verify signatures request as described in <a href="#">Request Format</a> . If a new signature was created ad hoc, the <b>fieldName</b> might not be returned.
	x	Int	X coordinate of the signature field.
	y	Int	Y coordinate of the signature field.
	width	Int	Signature field width.
	height	Int	Signature field height.
	pageNumber	Int	The page number on the document.
	dateFormat	String	The format string used to form the date string.
	timeformat	String	The format string used to form the time string.
	graphicalImage	Boolean	Whether the graphical image is displayed.

<b>Group</b>	<b>Name</b>	<b>Type</b>	<b>Description</b>
	signer	Boolean	Whether the signer name is displayed.
	date	Boolean	Whether the date is displayed.
	showTitle	Boolean	Whether the title is displayed.
	showReason	Boolean	Whether the Reason is displayed.
	title	String	The signer title.
	reason	String	The selected sign reason or reject reason.
<b>Document</b>	contentType	String	The document content-type (PDF).
	content	String	The content, in base64 format.

## Verifying Signatures in a File

The Verify Signatures operation enables your web application to validate signatures in a document. The document types supported for document verification are PDF, DOCX and XLSX.

### ***Request Format***

To initiate the verification operation, an XML request is sent to:

```
https://<host name>/Sign/VerifyServiceXML.
```

The HTTP header of the request must specify the POST method, and it must include a Content-Type parameter with a value “**application/x-www-form-urlencoded**”. The actual data of the HTTP-POST command should include the parameter “**inputXML**”, and the contents should be an XML structure. The XML structure must contain the signed file to be verified, Base64 encoded. For a full description of the parameters in the XML structure, refer to [Request Parameters](#).

### ***Request Parameters***

The request contains the file to verify and its type. For an example of such an XML structure, refer to [Sample XML Request](#).

Group	Name	Type	Description
<b>Document</b> This group is required	content	String	A Base64 encoded file to verify.
	contentType	String	The type of document to verify. Supported types: <b>PDF, Docx, Xlsx</b> .

### ***Response Format***

The Response XML will be returned by the Web Agent to your web app. Either a success or an error response will be sent.

In case of an error, the returnCode parameter will be -1. In addition, the errorMessage parameter will contain the detailed error message.

If the operation was successfully completed, an XML file will be returned, containing the validation parameters for each field in the document as well as a general validation code.

For an example of the XML structure sent upon success, refer to [Sample Response](#).

Group	Name	Type	Description	Notes
<b>Status</b>	validationStatus	Int	The validation status of the entire document. The possible values are:  0 – Valid. All signatures in the document are valid. There are no unsigned signature fields or invalid signatures, 1 – Invalid. There is at least one invalid signature in the document. 2 – Error. The signature validation process failed. Signature validity cannot be determined. 3 – Incomplete. At least one unsigned field exists in the document and all other signatures (if exist) are valid. If there is at least one invalid signature, Invalid status (1) will be returned. 4 – Unknown. The status of at least one signature is ‘unknown’ (see <a href="#">status</a> below). 5 – Empty. The Document does not contain any signature fields.	
	returnCode	Int	Specifies an internal return code that was returned when trying to verify signatures in a document. Will return only in case of an error. For a description of the error codes, refer to <a href="#">CoSign Signature Local – Signing/Verifying Error</a> Messages.	
	errorMessage	String	A textual error message. This message will be returned only in case of an error.	
<b>Field</b>  One or more occurrences within the XML	fieldname	String	The name of the signature field that was verified.	Multiple instances
	status	String	Validity status of a specific signature field. The possible values are:  0 – valid 1 – invalid 2 – not signed 3 – unknown	
	signingTime	String	Time and Date of the signing event.  The GMT offset will be taken from the machine that runs the Web App application. The format is: dd/MM/yyyy HH:mm:ssZ	
	signFileTime	FILETIME	Date and Time of the signing event in FILETIME format	
	signerName	String	The name of the signer (Common Name).	
	signReason	String	Signing Reason.	

Group	Name	Type	Description	Notes
	certErrorStatus	Int	The certificate's status. The possible values are: 0 – The certificate status was not checked. 1 – The certificate is OK. 2 – The certificate is revoked. 4 – The certificate chain is not complete because one of the certificates in the chain is not trusted. 8 – The revocation status of the certificate or one of the certificates in the certificate chain is unknown.	

## XML and Redirection URL Samples

This section provides samples of XML files and Redirection URLs participating in the various actions (requests and responses) sent from your web app to the Web Agent and received back.

### Uploading a File and Initiating Signing – Samples

The following section provide Upload File and Initiate Signing samples.

#### *Sample Protocol Request*

The POST request uses urlencoding for the payload body:

```
POST http://foo.bar.com/Sign/UploadFileToSign HTTP/1.1
Content-Type: application/x-www-form-urlencoded
Host: foo.bar.com
Content-Length: 166173
Expect: 100-continue
Connection: Keep-Alive

inputXML=%3c%3fxml+version%3d%221.0%22+encoding%3d%22utf-8%22 ...
```

Note that %3c is url encoded “<”, %3f is “?”, etc.

#### *Sample XML Request*

```
<?xml version="1.0" encoding="utf-8" ?>
<request>
  <SigField>
    <fieldNameToSign>SampleField</fieldNameToSign>
    <enforceFieldToSign>>false</enforceFieldToSign>
  </SigField>
<SigProfiles>
  <SigProfile>
    <fieldName>SampleField</fieldName>
    <x>370</x>
    <y>120</y>
    <width>165</width>
    <height>55</height>
    <pageNumber>-1</pageNumber>
    <title>CEO</title>
    <showTitle>>true</showTitle>
    <dateFormat>MM/dd/yyyy</dateFormat>
    <timeFormat>hh:mm tt</timeFormat>
    <signer>>true</signer>
    <date>>true</date>
    <graphicalImage>>true</graphicalImage>
    <isNewFieldDefaults>>false</isNewFieldDefaults>
  </SigProfile>
</SigProfiles>
<SignReasons>
  <signReason>I am the author of the document</signReason>
  <signReason>I Approve the document</signReason>
</SignReasons>
<RejectReasons>
  <rejectReason>Not approved due to budgetary constraints</rejectReason>
  <rejectReason>Legal opinion required</rejectReason>
  <rejectReason>Management approval required</rejectReason>
</RejectReasons>
<Document>
  <content> <Base64 file> </content>
```

```

    <contentType> pdf </contentType>
    <fileID> 123as1d4 </fileID>
    <fileName>Getting Started.pdf</fileName>
  </Document>
  <Logic>
    <isWorkflowMode>true</isWorkflowMode>
    <allowAdHoc>true</allowAdHoc>
    <enforceReason>false</enforceReason>
    <allowUserReason>true</allowUserReason>
    <verify>false</verify>
  </Logic>
  <Url>
    <finishURL>http://example.com/finish.html</finishURL>
    <redirectIFrame>true</redirectIFrame>
  </Url>
  <Auth>
    <username>usernameExample</username>
    <password>passwordExample</password>
    <keepUserLogin>false</keepUserLogin>
  </Auth>
  <Layout>
    <layoutMask>24</layoutMask>
  </Layout>
</request>

```

### **Sample XML Response**

```

<?xml version="1.0" encoding="utf-8"?>
<response xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:xsi="http://www.w3.
org/2001/XMLSchema-instance" type="error">
  <Error>
    <returnCode>0</returnCode>
    <errorMessage />
  </Error>
  <Session>
    <sessionId>307725171</sessionId>
    <docId>123as1d4 </docId>
  </Session>
</response>

```

### **Redirecting a User to Web Agent – Sample**

```
http://cosign-web-app.org.com/Sign/SignCeremony?sessionId=307725171
```

### **Redirecting User Back to your Web Application – Sample**

```
https://webapp.org.com/?sessionId=307725171&docId=123as1d4&returnCode=0
```

### **Downloading a Signed File – Samples**

#### **Sample Request**

```
Sign/DownloadSignedFileG?sessionId =12321_3213213
```

#### **Sample Response**

```

<?xml version="1.0" encoding="utf-8"?>
<response type="signing">
  <Error>
    <returnCode>0</returnCode>
  </Error>
  <Session>
    <sessionId>swaSes_84093_78193977</sessionId>

```

```

    <docId>84093</docId>
  </Session>
  <Result>
    <operationStatus>1</operationStatus>
  </Result>
  <SigDetails>
    <fieldName>SampleField</fieldName>
    <x>367</x>
    <y>125</y>
    <width>159</width>
    <height>61</height>
    <pageNumber>1</pageNumber>
    <dateFormat>dd MMM yyyy</dateFormat>
    <timeformat>hh:mm tt</timeformat>
    <graphicalImage>True</graphicalImage>
    <signer>False</signer>
    <date>False</date>
    <showTitle>True</showTitle>
    <showReason>True</showReason>
    <title>CEO</title>
    <reason>I am the author of the document</reason>
  </SigDetails>
</response>

```

## Uploading a File and Verifying Signatures – Samples

### *Sample Request*

```

<?xml version="1.0" encoding="utf-8" ?>
<request>
  <Document>
    <content> <Base64 file> </content>
    <contentType> pdf </contentType>
  </Document>
</request>

```

### *Sample Response*

```

<?xml version="1.0" encoding="utf-8"?>
<response type="verify">
  <Verify>
    <Status>
      <validationStatus>0</validationStatus>
    </Status>
    <Fields>
      <Field>
        <fieldName>SampleField</fieldName>
        <signingTime>10/08/2012 15:45:39</signingTime>
        <signerName>John Miller</signerName>
      <signReason>I approve this document</signReason>
      <signerEmail>johnm@arx.com</signerEmail>
      <status>0</status>
      <certErrorStatus>1</certErrorStatus>
    </Field>
  </Fields>
</Verify>
</response>

```



## Appendix A: Web Agent API Error Codes

This appendix describes the error codes returned by the Web Agent API.

### Convert to PDF Error Codes

Error Code	Description
601	Failed to convert the document to PDF format.
604	File type is unsupported. Supported file types are: pdf, doc, docx, xsl, and xlsx.
605	Document to convert exceeds the maximum number of pages (200 by default).

### API Request Error Codes

Error Code	Description
706	Failed to handle a sign request.
707	The request has timed out.
708	Failed to build a response.
709	Signature Profile has an invalid page number.
710	The document does not contain any signatures fields and the user is not allowed to create new fields.
711	Signature Field to Sign does not match any fields in the document (including Signature Profile).
712	API request denied because the Server is not allowed to handle the API request. This is controlled by a configurable web.config parameter (refer to <a href="#">Setting Basic Integration Settings in the Web App Configuration File</a> ).
713	A Signature Profile with the same name already exists in the document.
714	The request does not contain a Signature Field to Sign name.
715	The user credentials given in the XML are incorrect.
716	The mandatory finishURL element is missing.
717	“Point of Sale” mode is not supported for Office files (docx, xlsx).
718	Signature profiles are not supported for Office files (docx, xlsx).
720	Signature profile has an invalid width.
721	Signature profile has an invalid height.
722	Signature profile has an invalid field name. Field name is limited to 50 characters and supports only alphanumeric characters, +, -, =, _

<b>Error Code</b>	<b>Description</b>
723	POS mode does not support suggesting a field name to sign.
724	POS mode does not support enforcing signing of a specific field.
725	POS mode does not support the display of a Reject button.