Digital signatures for SAP applications using the Digital Signature Tool – Developer Guideline and best practices

Applies to:

Class-based digital signature tool (SAP application component CA-DSG).

Summary

This document describes how to implement the digital signature functionality with the Digital Signature Tool of SAP. It contains information about necessary master data and customizing settings, as well as tips and tricks for the application integration.

Author: Dr. Uwe Dittes

Company: SAP AG

Created on: 19 January 2007

Author Bio

Dr. Uwe Dittes made his PhD in Chemistry at the University of Heidelberg. After working several years in the chemical and pharmaceutical industry (development & production), he joined the development organisation of SAP in 2001. He has a strong expertise in SAP PP-PI (especially master recipe, production versions, electronic batch record [EBR], execution steps [XSteps]) and xMII content development.

Table of Contents

General remarks:

- This best practice document is based on the implementation guide (version 3.0) for the Digital Signature Tool. The implementation guide is part of the SAP note 700495 and could be updated without further notice.
- Please also check for corrections for the digital signature application component (CA-DSG).
- If you need additional help for the implementation after reading this document, you should request SAP remote-consulting (see SAP note 83020).
- The complimentary Customer Support of SAP will not support errors that are caused by modifications of the SAP standard system or by customer development (see SAP note 7).

Implement the Guide Digital Signature-Tool (CA-DSG)

Abbreviations:

TA = transaction

Prerequisites:

- SAP Basis Release 6.20
- Good knowledge about the functionality of the Digital Signature-Tool
- Advanced experience in ABAP- and ABAP-Object-programming
- Good knowledge about the application you want to implement the signature tool

Use of the Digital Signature Tool

Since release 4.0A, the SAP system has provided digital signatures, to secure digital documents and business processes. Previously, such digital signatures were only available for selected positions in the SAP system. If a function was to be connected to further applications, the program code had to be extended according to the application. This is very time consuming.

As of release 6.20, a standardized and flexible programming interface is available, with which the digital signature can be connected to any areas in both R/3 systems and other systems, such as APO, CRM, etc. In principle, no changes are made to the existing functions, and the following functions are added:

The document to be signed is called up by the relevant application and forwarded to the signature tool. When the signature function is called up, the default values in the application decide if the document to be signed should be managed by the application itself or by the signature function.

For signature strategies with several individual signatures, it is possible to run verification on executed signatures to check their validity. The verification is independent of whether or not the document is managed by the signature function or the application.

The user has the option of displaying the document before the signature operation. (*However, this is currently only available for .txt type files.*)

Effects on Customizing

To implement the digital signature for the required applications, you must perform the following activities in the Customizing, after the upgrade:

- Create the general user settings (see Defining Basis Settings for Digital Signature).

- If you want to use the signatures without a signature strategy, create the corresponding settings (see Specify Signature Method for Approval using Simple Signature).
- If you want to work with signature strategies, you must define authorization groups, individual signatures and a signature strategy.

Further notes

For more information on the digital signature, see the following release information:

- Digital Signature (for Release 4.0A)
- Digital Signatures (for Release 4.5A)
- Changes to the Digital Signature (for Release 4.6C)

You will get additional information, examples and technical details in the class documentation of the corresponding classes and interfaces of package DS (Digital Signature).

Processing Steps

To include digital signatures in the application the following steps must be carried out:

- Define a log structure and a database table in the Data Dictionary
- Enter a sub object for the application log (TA SLG0)
- Enter the application and the signature object in the system tables (Transport request!)
- Include the programming interface in the program of the application
- Make the following entries in Customizing:
    - Define an authorization group (TA ELSIG01),
    - Define signature single steps and assignment of the single steps to the authorization group (TA ELSIG02)
    - Define a signature strategy (Assignment of the single steps to the strategy, definition of the sequence and release statuses) (TA ELSIG00)
    - Make settings for the signature object for cases in which there is no signature strategy (TA ELSIG03N)

If needed then implement the BADI interfaces IF_EX_DS_AUTHORITY and IF_EX_DS_CONTEXT.

Enter the authorization group in the authorization profile (User maintenance)

System Tables

The maintenance views for the application and signature objects are available for system tables. **You always define one application. Multiple signature objects can be assigned to this application**.

Definition of the Application (TA SIGNA):

Application	Description
8-figure abbreviation	Short text

Definition of the Signature Object (TA SIGNO):

Application	Object	Meta Data	Log Structure	Sub Object
8-figure abbreviation	8-figure abbreviation	Name of a meta data table	Name of a DDIC structure	Sub object corresponds to the application
Comment	**Observation**	**Object Description**	**Document**	**Name**
Required Allowed Not allowed	Required Allowed Not allowed	Allowed Not allowed	Required Allowed Not allowed	Short text

Observations:

The client (data element MANDT) and signature ID (data element SIGN_GUID_22) are obligatory key fields in the metadata table. If you use change documents you must include them as attributes. In addition, the attributes of the metadata table should uniquely identify one object of the application.

When you define the log structure you should note that the fields and their data elements must also be included in the metadata table. However, not all the metadata table fields have to be in the log structure. The first entry must be the include SIGN_PROT_STRUC. The fields SIGN_GUID_22 and MANDT should not be included.

In the log structure a total width of 250 characters is allowed. Examples for log structures of applications that include *SIG_PROT_STRUC* can be found via transaction SE11: Enter the data type *SIGN_PROT_STRUC* and execute the Where-Used List by setting the flag named *'Used in Structures'*.

Register the sub object using transaction SLG0. The entry is made under the object *CDSG1*, which is necessary for logging signature actions in the application log:

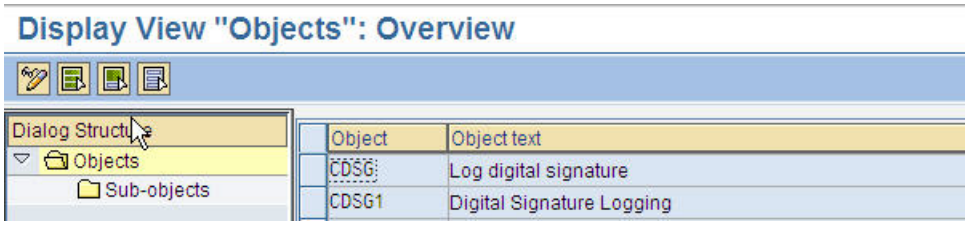


Figure 1: application log registration using transaction SLG0

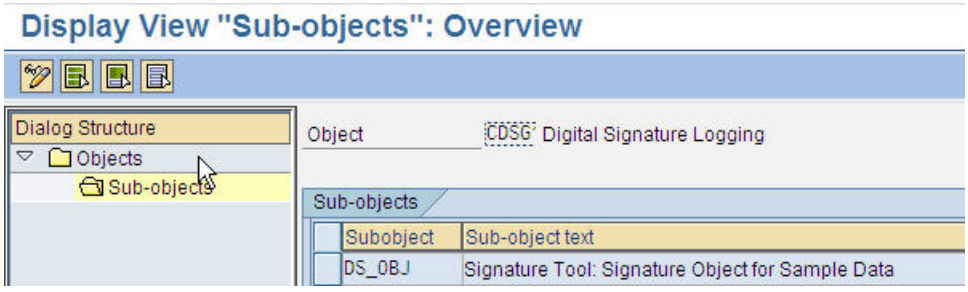


Figure 2: Define sub object for object CDSG1 (transaction SLG0)

When you define a signature object in transaction SIGNO, all fields should contain a consolidated setting since each change is saved in a version and the historical versions are delivered as well.

Metadata tables and log structures that have already been delivered must not be changed. They can be replaced by new ones in SIGNO, but to guarantee a correct history they must remain in the Dictionary.

Use F1 help for further information about each field.

Package interface DS_API in package DS

All the DDIC objects and methods to be used are described in the package interface DS_API. The following interface methods can be used:

Class	Method	Description
CL_DS_RUNTIME	GET_INSTANCE	Generate instance of the runtime (singleton)
Interface	**Method**	**Description**
IF_DS_RUNTIME	CREATE	Generate instance of the signature process
	GET_BY_ID	Generate instance of an existing signature process using the signature ID
	GET_BY_META	Generate instance when specifying metadata
	GET_REG_INFO	Read registration info for the applications
	REFRESH_BY_LIST	Update signature data
	GET_DATA_BY_LIST	Read signature data as list
	FREE	Delete all instances
IF_DS_SIGN	SIGN	Execute digital signature for a document
	GET_STEPS	Read data for the individual step for the signature strategy
	GET_STEPS_WITH_DOC	Returns Saved Data/Documents for Signature Step
	GET_STEPS_WITH_COMMENT	Delivers Saved Data and Comment for Signature Step
	SAVE	Save data for signature
	UNDO	Discard signature data
	UNLOCK	Unlock Header Data Record for Signature
	GET_ID	Read signature ID
	REFRESH_SIGNATURE	Update signature data
	FREE	Delete instance
	GET_STATE	Determine signature status
	GET_META_BY_ID	Read metadata using the signature ID
IF_EX_DS_AUTHORITY	AUTHORITY_CHECK	Authorization check
IF_EX_DS_CONTEXT	GET_DOCUMENT	Generate document to be signed

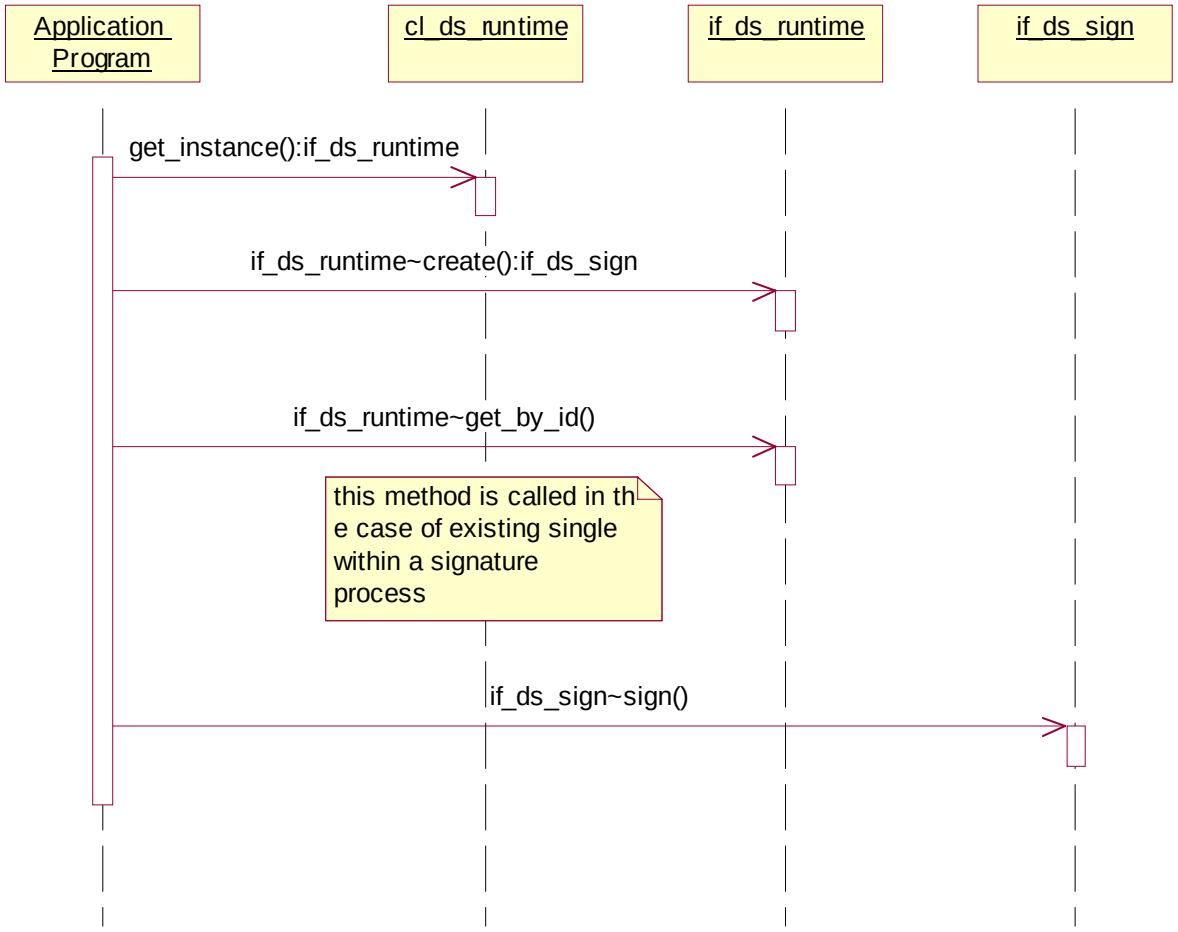
Remarks on the package interface:

- All methods and interfaces are documented in the system.
- The methods SAVE and UNDO of the interface IF_DS_SIGN should only be used if the signature data is not automatically saved (See parameters of the method GET_INSTANCE of the class CL_DS_RUNTIME).
- Some applications (e. g. document management, log books) need to hold and store their documents in their own application database tables. Those documents aren't stored in the corresponding document database table of the signature tool. Therefore the BADI method GET_DOCUMENT must be implemented so the document could be handed over from the application to the signature tool while signing.

- If the authorization check (BADI Method AUTHORITY_CHECK) is not implemented a standard authorization check according to the authorization object C_SIGN is carried out.
- The interface IF_DS_SIGN contains additional methods that were created during corrections. Check that the methods exist in your system or implement the necessary SAP notes:
  - Method GET_STEPS_WITH_DOC (SAP note 827471)
  - Method GET_STEPS_WITH_COMMENT (SAP note 992731)
  - Method UNLOCK (SAP note 951699)

Sequence diagram for implementation of the signature functionality

The call of the signature tool in the program might look as follows:



Customizing settings for the Digital Signature tool

Screenshot 'Define Signature Object' (Transaction ELSIG03N):

It is necessary to define the signature method for your signature object (TC ELSIG03N):

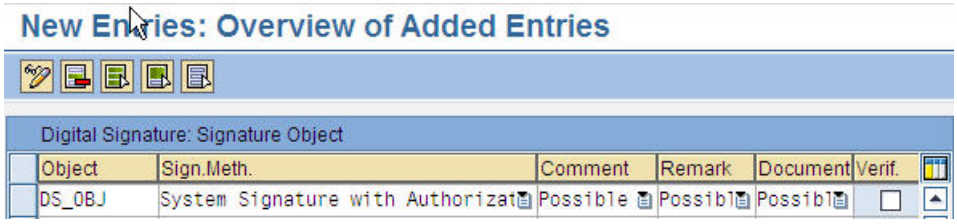


Figure 3: Signature method for signature object (transaction ELSIG03N)

If your application doesn't transfer a signature strategy during creation of the signature process, this customizing setting is taken into account. Only one signature is necessary to complete the signature process ('two eyes check').

The signature method 'System signature with Authorization by SAP User-ID/Password' is commonly used by most applications. The user has to enter his/her SAP password during signing. The two other signature methods 'User signatures with External Security product with/without verification' need additional hardware like Smartcard reader etc. Please contact SAP to get actual information about supported hardware.

Multi-user signing using signature strategies

For approval procedures using digital signatures more than one user has to sign successfully before the approval (signature process) is confirmed, e. g. "four eye cross checks" etc. SAP supports multi-user signing with the help of signature strategies. Signature strategies are defined in the customizing of the Digital signature tool (see previous chapter). They consist of the following information (TA ELSIG00):

- How many different users must have signed successfully to complete a signature process (number of individual signatures)
- In which order the individual signatures must be executed (signature sequence)
- Which of the defined individual signatures are necessary to complete the current signature process (release status)

Remark:

In most cases two to four individual signatures per signature strategies are sufficient. If more than eight individual signatures are assigned there are technical restrictions concerning the maintenance of the signature sequence and the release status: See SAP notes 860115 & 861337 for further details.

Each SAP application that implements the Digital Signature tool could use the signature strategies that are maintained centrally.

The screenshots below illustrate the necessary actions to create an own signature strategy:

Screenshot "Define Authorization Group" (TA ELSIG01)

Authorization groups could be used if individual signatures from different user groups (worker, shift leader, QM, plant manager etc.) should be included in the signature strategy. Before an individual signature can be executed the system checks if the user belongs to the correct authorization group of the individual signature.

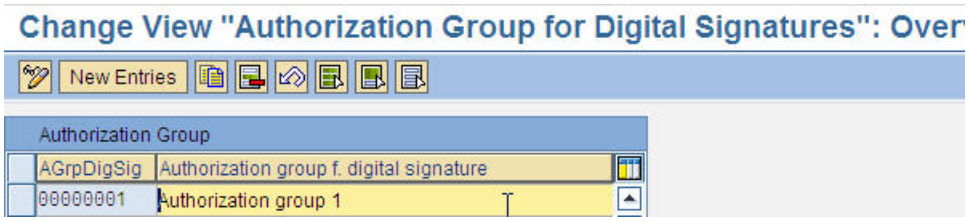


Figure 4: Define Authorization Group (Transaction ELSIG01)

Screenshot "Define Individual signatures" (TA ELSIG02)

Individual signatures are the digital abstraction for 'signing your name on a paper'. Before signature strategies can be created the necessary number of different individual signatures must be created first:

Figure 5: Define Individual Signatures (Transaction ELSIG02)

Screenshot 'Define Signature Strategy' (Transaction ELSIG00):

After creating the needed numbers of authorization groups and individual signatures the signature strategies could be defined:

- Name a signature strategy and add a helpful description
- Choose the signature method you need
- Decide, if comments, remarks or and/or documents should be displayed (options: possible, disallowed, or required). Depending on the settings,
  - The signatory is either allowed or not allowed to enter text comments that can be edited freely for the signature to be executed. If a comment is required, the user can only execute the digital signature if they have first created a comment.
  - The signatory is either allowed or not allowed to add a remark from a list of predefined remarks, to the signature that has been executed. If a remark is required, the user can only execute the digital signature once they have selected a remark. The existence of remarks depends on the relevant application.
  - The signatory may be required, allowed or forbidden to view the document to be signed.

- Set the flag for verification, if necessary:
  - Depending on the settings, it may be possible to verify all previously executed signatures when executing the digital signature.

Figure 6: Overview of signature strategies

Screenshot 'Assign individual signatures' (Transaction ELSIG00):

Depending on the numbers of different users that are taking part in the signature process the appropriate individual signatures should be assigned:

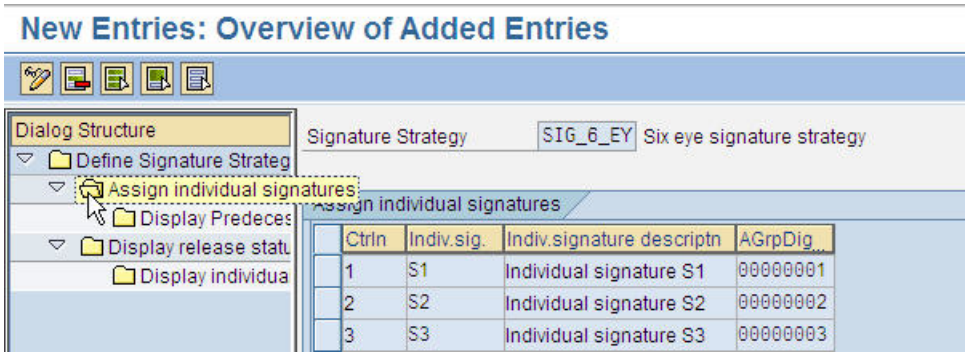


Figure 7: Assigned individual signatures for marked signature strategy

Remark:

Avoid it to assign one individual signature more than one time to the signature strategy. This could lead to confusion, e. g. during definition of release status and signature sequence definition.

Afterwards it is possible to define the sequence of the individual signatures and the release status for the signature strategy.

Screenshots 'Define signature sequence (Transaction ELSIG00):

Mark the signature strategy you want to maintain and press the button:

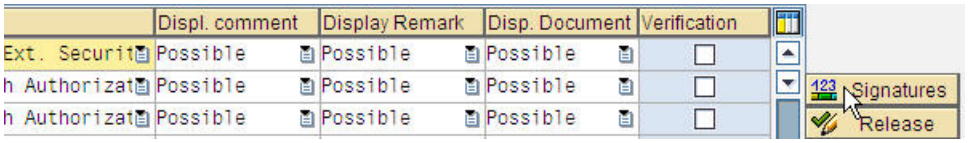


Figure 8: Maintain the sequence of individual signatures

A popup appears where you can define relationships for successors and predecessors:

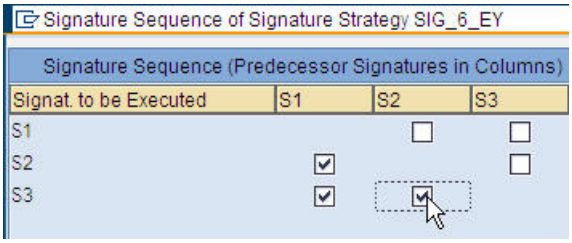


Figure 9: Example for a signature sequence

In this example the individual signature S1 is the first signature. If S1 was executed successfully, the individual signature S2 can be executed, followed by the individual signature S3.

Remark:

If no signature sequence is defined the system will offer all individual signatures for selection (dropdown box for the authorization group). The system also checks if the signatory belongs to the required authorization group of the chosen individual signature.

Screenshots 'Define release status (Transaction ELSIG00):

Mark the signature strategy you want to maintain and press the button 'Release Status':

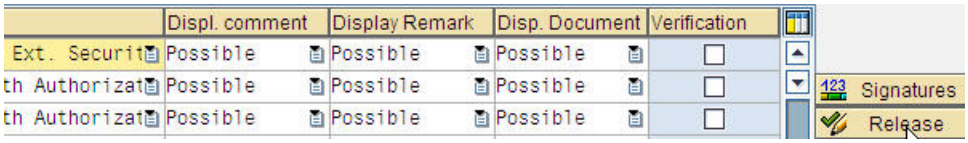


Figure 10: Maintain the release status for a marked signature strategy

A popup appears that shows the valid combinations of release status:

Figure 11: Example for a release status

In this example the individual signatures S1, S2 and S3 must be executed successfully before the corresponding signature process is finished and released.

Remark:

If no release status is marked the system checks that all individual signatures assigned to the signature strategy must be successfully executed before the signature process is finished.

Assign signature strategies via customizing settings in applications

At the beginning of a signing procedure the digital signature tool requires the information which signature strategy – if any – should be used (to create a signature instance). Therefore the application that implemented the digital signature tool has to provide suited customizing settings. There exist several examples in the SAP standard how the application customizing could look like, e. g.:

- Logbook: Dig. signature customizing (TA DIACLC3) [PLM-LBK{EA-APPL Layer}]
- Assign control recipe destination (TA O10C) [PP-PI-PMA-RCP {EA-APPL Layer}]
- Batch record: Overall profile (TA COCS) [PP-PI-PDO {SAP_APPL Layer}]

Remark:

Depending on your system configuration (extension switch setting [activation of EA-APPL layer]) you don't have all of the above listed transactions available.

Example for the implementation of a multi-user scenario

This section provides an example how an implementation of a multi-user scenario using signature strategies could look like. The signature tool is implemented for a (virtual) application using local objects (package $TMP, customer namespace [Z coding]. In addition to the comments made in this chapter you will find valuable information in the source coding passages of the example.

Virtual sample application

The application creates bookings that have a unique ID. An additional text could be added. It is possible to release or cancel the booking. It is necessary to confirm the release or the cancelling of the booking by executing a digital signature. The usage of signature strategies is recommended.

Necessary DDIC data (customer namespace)

Define a log structure, a Meta database table and a database table for the application data in the Data Dictionary: Create the following data dictionary objects:

Database table ZDS_BOOKING_EX (Application data)

- Short Description: Sample database table for digital signature implementation
- Delivery class: A (application and master data)
- Enhancement category: Cannot be enhanced

Field	Key	Data element	Data type	Length	Decimals
MANDT	X		CLNT	3	0
DOC_ID	X	DOKNR	CHAR	25	0
BOOKING_TXT		TEXT40	CHAR	40	0
BOOKING_STATUS			CHAR	1	0

Use predefined data types for the fields MANDT and BOOKING_STATUS

Technical settings for ZDS_BOOKING_EX:

- Data class: APPL0 (Master data, transparent table)
- Size category: 0 (0 – 8800)

Database table ZDS_META_EXAMPLE:

- Short Description: Meta data table for digital signature implementation
- Delivery class: A (application and master data)
- Enhancement category: Cannot be enhanced

Field	Key	Data element	Data type	Length	Decimals
MANDT	X		CLNT	3	0
SIGNID	X	SIGN_GUID_22	CHAR	22	0
DOC_ID		DOKNR	CHAR	25	0

Technical settings for ZDS_META_EXAMPLE:

- Data class: APPL0 (Master data, transparent table)
- Size category: 0 (0 – 8800)

Structure ZDS_EXAMPLE_SIGN_LOG:

- Short Description: Protocol log structure for digital signature implementation
- Enhancement category: Cannot be enhanced

Component	RType	Component type	Data type	Length	Decimals
.INCLUDE		SIGN_PROT_STRUC			
DOC_ID		DOKNR	CHAR	25	0

Insert the include SIGN_PROT_STRUC at first position (see section ‘system tables’)

Additional sub object for the application log (TA SLG0)

Call transaction SLG0 to create an additional sub object for the digital signature log. Choose object ‘CDSG1’ (Digital Signature Logging).

Check if an entry for the sub object DS_OBJ (Sub-object text: Signature Object for Example Data for Signature Tool) exists.

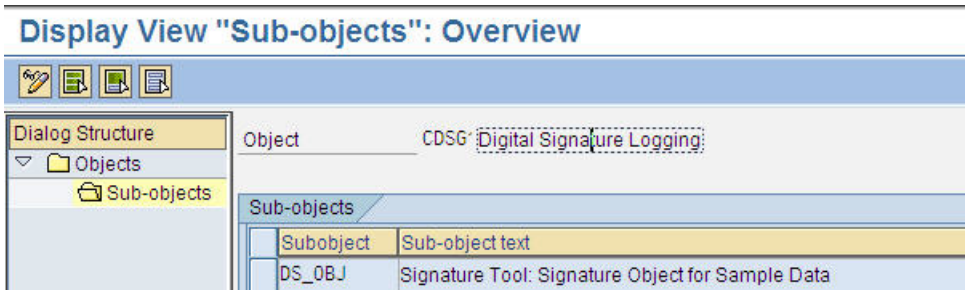


Figure 12: Definition of a sub object for the application log

Remark:

The sub object text determines the field value 'Reason for Signature' shown in the digital signature log (transaction DSAL) for the corresponding log message.

Registration of new application and its signature object

Enter the application and the signature object in the system tables:

Register application (TA: SIGNA):

Check if the following entry exists or create a new entry:

- Application: DS_EXAMP
- Description: Example Application for Signature Tool

Figure 13: Registration of sample application

Register signature object (TA SIGNO):

Modify the following entry or create a new entry:

- Application: DS_EXAMP
- Object: DS_OBJ
- Meta table: ZDS_META_EXAMPLE
- Log structure: ZDS_EXAMPLE_SIGN_LOG
- Sub object: DS_OBJ

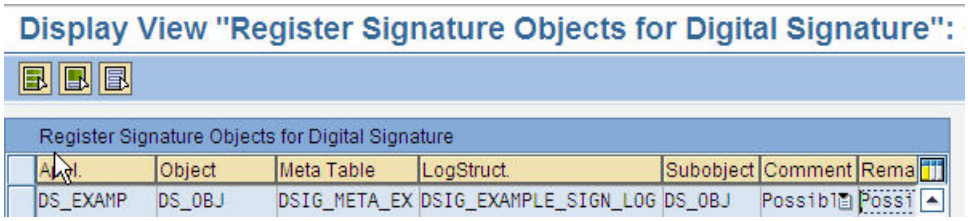


Figure 14: Register the signature object

- Comment: Possible Remark: Possible Object description: 'X' Document: Possible
- Description: Signature Object for Example Data for Signature Tool

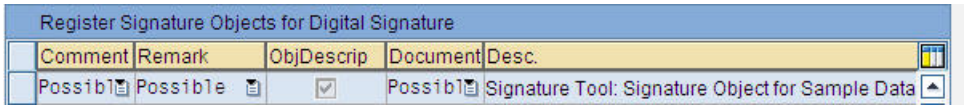


Figure 15: Register the signature object (continued)

Remark:

If the settings for comment, remark and document defined for the signature object are set to 'disallowed' they can't be overruled by the corresponding settings of the signature strategy used for signing the object.

Implementation of the signature tool for the virtual application

Include the programming interface in the program of the application. Create the local program ZDS_BOOKING_EXAMPLE as an executable program:

- Title: Sample report for digital signature implementation

Copy the source coding from section 'appendix'.

Maintain the selection texts of the report (menu path [SE38/SE80]: Go to -> Text Elements -> Selection Texts):

Name	Text	Dictionary
P_DOCTXT	Short text for document	
P_DOC_ID	Document	X
P_DS_TYP	Type of Signature	X
P_SAVEDB	Save changes on database	
P_SIGNER	Signatory	X
P_SI_STR	Signature Strategy	X
P_STATUS	Status of the document file	

Customizing settings for signature strategies

Make the following entries in Customizing:

- Define an authorization group
  - Call transaction ELSIG01 and create a new authorization group:
    - Authorization group: ZDSGROUP
    - Description: Sample authorization group (DS example)
- Define individual signatures and assign them to the authorization group
  - Call transaction ELSIG02 and create new entries for the individual signatures you want to use in a signature strategy.

Individual signature	Authorization group	Individual signature description
ZDS_SIG1	ZDSGROUP	Signer 1 (DS example)
ZDS_SIG2	ZDSGROUP	Signer 2 (DS example)

- Define a signature strategy: Assign the individual signatures to the strategy, define sequence and set release status, if necessary:
  - A signature strategy containing two signers is defined via TA ELSIG00:
    - Signature strategy: ZDS_STRA
    - Description: Signature strategy (DS example)
    - Signature method: System signature with Authorization by SAP User ID/Password
  - Mark the new entry for the signature strategy and assign the individual signatures ZDS_SIG1 & ZDS_SIG2:
    - Go back to the overview screen of the signature strategy maintenance. Mark the signature strategy ZDS_STRA and press the buttons to maintain the signature sequence and release statuses

- Make settings for the signature object for cases in which there is no signature strategy:
  - Call transaction ELSIG03N and enter the data:
    - Object: DS_OBJ
    - Signature method: System signature with Authorization by SAP User ID/Password

Create signatures for the virtual application

Start the report, enter values for the selection criteria and sign the sample files.

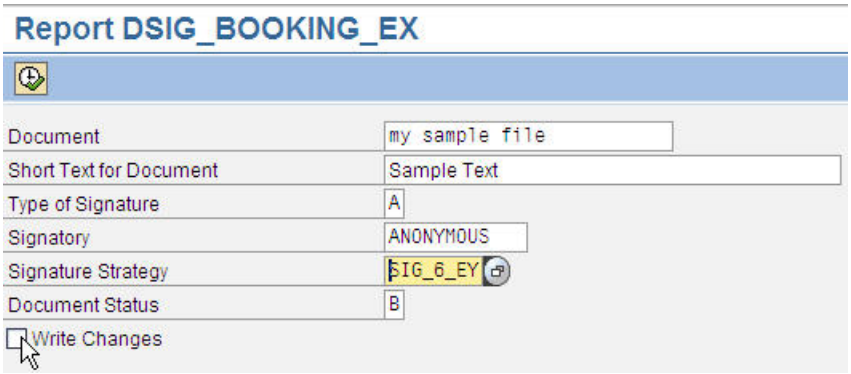


Figure 16: Selection criteria for sample report

Display the signature logs (transaction DSAL)

If you execute ZDS_BOOKING_EXAMPLE you will create log messages that could be displayed using transaction DSAL. Depending on the log structure ZDS_EXAMPLE_SIGN_LOG you will get the DOC_ID as selection criteria for the log entries:

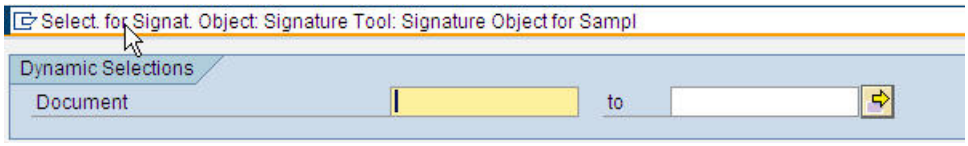


Figure 17: Dynamically generated selection criteria popup

Per default the values for the signature time are filled with the actual date:

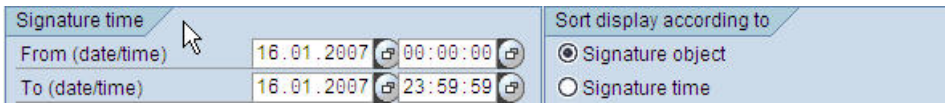


Figure 18: Default setting for the signature time

Attention should be paid for setting the proper time interval values.

If log entries for the selection criteria exist the result list is displayed (see excerpt):

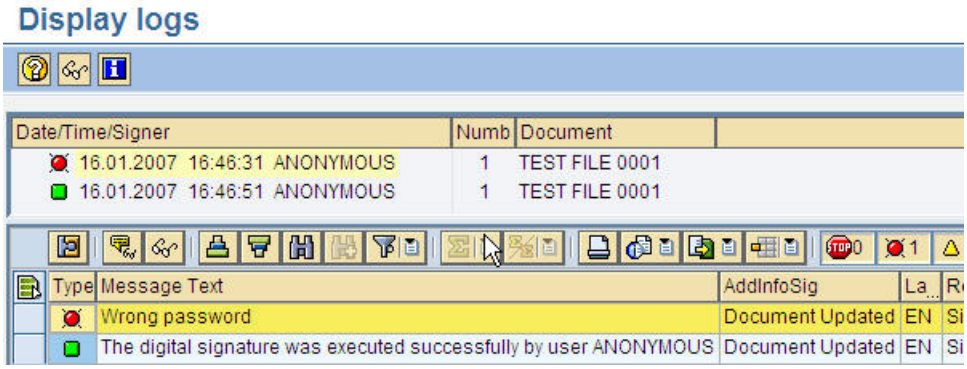


Figure 19: Found logs for sample application

Additional information

Remarks concerning the Digital Signature Log (TA DSAL)

- It is possible to add customer functions using the BADI CJ_DSAL_FUNC. The SAP notes 593041 (DSAL: Logs on the Digital Signature – BADI) and 594010 (DSAL: Example implementations for BADI CJ_DSAL_FUNC) contain more information how to setup customer functions.
- It is possible to show the comments that were entered during signing a signature object. As a prerequisite SAP note 994317 (Displaying comments for individual signatures with DSAL) must be implemented in your system.
- For the output of documents that were transferred to the digital signature tool during signing the SAP note 827417 (Signature tool: Output of signed documents using DSAL) describes the requirements.
- Transaction DSAL belongs to the SAP_APPL layer (due to historical reasons). If you implement the signature tool to applications that belong to the SAP_ABA layer you need to create an own transaction that allows displaying the log files.

Remarks concerning the Digital Signature Tool

- Currently the digital signature screens and pop-ups can only be used with a SAP WinGUI and not with HTML GUI. For applications that should run on the SAP portal this restriction also exists (Workaround: Use WINGUI based launch pads for your sample applications).
- The digital signature tool can't support the exchange of digital signature data between different systems (B2B scenarios etc.).
- Unfortunately there is no connection of the SAP work flow with the digital signature tool until now.

Tips and tricks for implementations of the Digital Signature Tool

The sample report DSIG_BOOKING_EX contains examples that illustrate most of the topics that could rise up during implementation of the Digital Signature Tool (see SAP note 910238 for further details). The following chapters provide information and hints to enable the decision what features of the Digital Signature Tool should be implemented.

Criteria for the usage of the different signature types

The Digital Signature tool offers three different signature types for multi user signing via signature strategies:

- Asynchronous signature with fixed user
- Asynchronous signature with changeable user
- Synchronous signature (with changeable user)

It depends on the requirement of the application which signature type fits best:

Asynchronous signatures are suited for scenarios, where the leading application should save and terminate after execution of an individual signature of the signature strategy. So it is possible to continue and finish the current signature process at another point of time. They are also suited to support scenarios where the signatories are spread over the companies (local distance) and aren't available at the same time at the same place to complete the signature process. In most cases the application exits after execution of the individual signature. The next signatory has then to start the application again to continue the signature process by signing.

Asynchronous signatures with fixed user set the logged system-user as a default user for signing (could be overruled by the implementation). After the system user signed successfully the next signatory is forced to log on to the SAP system and start the application in a new session before the signature process can be continued.

Remark:

When you use multilevel asynchronous signature processes with changeable users, the leading application should terminate after updating the data records. The lock is then reset automatically to the header record of the signature process. As long as the leading application is not terminated, the lock on the header record of the relevant signature process remains for safety reasons. This system response is correct.

If the application is not to be terminated after the updating of the first data record, the lock on the header record must be cancelled with the help of the UNLOCK method (interface IF_DS_SIGN). This new required method was implemented via SAP note 951699 (Locking problem w/ asynchronous signature w/ changeable user).

Synchronous signatures don't allow a break in a current signature process. The synchronous signature type could be used if all signatories are available at the same time at the same place to finish the current signature process (no local distance).

Remark:

During creation of a signature process (method IF_DS_RUNTIME~CREATE) the signature type is set by the leading application.

Implementation of multilevel signature processes versus simple signature processes

Simple signature processes consist of one individual signature. The signatory has two options: To sign or to sign not (cancel the individual signature tool). The corresponding signature process is finished when the signatory successfully signed.

Multilevel signature processes using signature strategies consist of more than one individual signature. Only if all required individual signatures are executed successfully the corresponding signature process is finished. Because more than one signatory has to sign, additional situations are possible, after the first signatory signed: The corresponding signature process is isn't finished and remains in progress. The next signatory now could either sign or cancel signing (the assigned individual signature). In addition the complete signature process can be cancelled: The current signature process gets the status 'cancelled'. If the application then wants to sign the signature object again a new signature process must be created. The signatories that successfully signed during the previous signature process then have to sign again in the new signature process.

Status	Signature status	Simple signature	Multilevel signature process
New Signature Process		Possible	Possible
Completed Signature Process	1	Possible	Possible
Canceled Signature Process	2	Impossible	Possible
Current Signature Process	3	Impossible	Possible

Remark:

The interface method IF_DS_SIGN~GET_STATUS (Determine Signature Status) should be used to get the actual status of a signature process.

The leading application that implemented the Digital Signature Tool should show different reactions based on the user input. It should be avoided that data of the signature object can be changed or modified as long as the corresponding signature process isn't finished.

Multilevel signature processes are often used to support status changes and/or approval procedures triggered by the leading application. Here it is very important to implement the appropriate application logic especially for signature processes that are in progress or cancelled.

The sample report DSIG_BOOKING_EX contains comments in the source coding passages that describe how the leading application could react based on the current status of a signature process.

As an example *application status A* should change to *application status B* after successful execution of the signature process. Here's an overview about the main activities the application has to implement:

- Get a signature process instance (by meta data of the application)
  - Determine the actual status of the signature process instance (IF_DS_SIGN~GET_STATUS)
  - Sign
    - User action: Signatory cancelled assigned individual signature (raised exception: `cx_ds_escape`)
      - Leading application catches the exception and triggers follow-up activities. There is no application status change at all.
    - User action: Signatory signed or cancelled the signature process
  - Determine the actual status of the signature process instance (IF_DS_SIGN~GET_STATUS)
    - Signature process completed
      - Leading application continues with their follow-up activities (successful execution of the signature process), *application status B* is set.
    - Signature process in process (only for asynchronous signature processes)
      - Leading application sets intermediate *application status D* forcing the users to complete the signature process first.
    - Signature process cancelled
      - Leading application should react on the cancellation: Either by setting a new *application status C* indicating the cancellation of the signature process. Or by resetting the application status to the previous *application status A*.

Appendix: Source code of report ZDS_BOOKING_EXAMPLE

```
*&---------------------------------------------------------------------*
*& Report  DSIG_BOOKING_EX
*&
*&---------------------------------------------------------------------*
*& Sample report for the implementation of the signature tool to an
*& application (see note 700495)
*& Create the DDIC elements and customizing entries first
*&---------------------------------------------------------------------*

REPORT  dsig_booking_ex.

PARAMETERS:
p_doc_id TYPE doknr,
p_doctxt TYPE text40 DEFAULT text-exa,
* Sign types:
* - Asynchronous with fixed user (A, default setting)
* - Asynchronous with changeable user (B)
* - Synchronous (with changeable user) (C)
p_ds_typ TYPE sign_type DEFAULT 'A',
* Signing user
p_signer TYPE signer DEFAULT sy-uname,
* Signature strategy
p_si_str TYPE signstrat MATCHCODE OBJECT sign_s_strategy,
* Status of the file ('B' = booked, 'C' = cancelled)
p_status TYPE c DEFAULT 'B',
* Flag to trigger update on database
p_savedb TYPE xfeld DEFAULT ''.

DATA:
lt_sign            TYPE sign_ref_tab,
lt_sign_remark     TYPE sign_remark_tab,
lw_sign_remark     TYPE sign_remark_struc,
ls_selected_remark TYPE sign_remark_struc,
lw_sign            TYPE sign_ref_struc,
ls_meta            TYPE dsig_meta_ex,
ls_dsig_booking_new TYPE dsig_booking_ex,
ls_dsig_booking_old TYPE dsig_booking_ex,
l_default_remark_id TYPE sign_remark_struc-id,
l_change_appl_data TYPE c VALUE '',
l_sign_state       TYPE sign_state,
l_document         TYPE string,
l_error_text       TYPE string,
l_error_ds         TYPE xfeld VALUE '',
l_signing_finished TYPE xfeld VALUE '',
lo_runtime         TYPE REF TO if_ds_runtime,
lo_sign_inst       TYPE REF TO if_ds_sign,
lo_cx_metadata     TYPE REF TO cx_ds_exception,
lo_cx_ds_escape    TYPE REF TO cx_ds_exception,
lo_cx_ds_message   TYPE REF TO cx_ds_message,
lo_cx_ds_exception TYPE REF TO cx_ds_exception.

CONSTANTS:
* Constants for the registration of the application and its
* signature object
co_ds_application  TYPE sign_appl   VALUE 'DS_EXAMP',
co_ds_signobject   TYPE sign_object VALUE 'DS_OBJ',
* Constants that describe the state of a signature process
co_state_sig_proc_new TYPE sign_state VALUE ' ',
co_state_sig_proc_closed TYPE sign_state VALUE '1',
co_state_sig_proc_canc TYPE sign_state VALUE '2',
co_state_sig_proc_open TYPE sign_state VALUE '3'.

************************************************************
* Application specific part
************************************************************

* Get data of the application object from the database
SELECT SINGLE * FROM dsig_booking_ex INTO ls_dsig_booking_old
  WHERE doc_id = p_doc_id.
IF ls_dsig_booking_old IS INITIAL.
* Create a new database entry
  l_change_appl_data = 'I'.
ENDIF.
* Transfer the entered parameter values
ls_dsig_booking_new-doc_id         = p_doc_id.
ls_dsig_booking_new-booking_txt    = p_doctxt.
ls_dsig_booking_new-booking_status = p_status.
```



```
* Only trigger signature and data update if the old data
* is different from the new one
IF ls_dsig_booking_new = ls_dsig_booking_old.
  WRITE:/ text-ncd.
  EXIT.
ENDIF.
IF NOT ls_dsig_booking_old IS INITIAL.
  l_change_appl_data = 'U'.
ENDIF.

************************************************************
* Implementation part of the signature tool
************************************************************

CLASS cl_ds_runtime DEFINITION LOAD.

* Get DS runtime object
lo_runtime = cl_ds_runtime=>get_instance( ).

* Fill the remark table
lw_sign_remark-id = '01'.
lw_sign_remark-sign_remark = text-dbk.
INSERT lw_sign_remark INTO TABLE lt_sign_remark.
* First remark should be set as the default remark in the
* remark list box of the signature popup screen
l_default_remark_id = lw_sign_remark-id.

lw_sign_remark-id = '02'.
lw_sign_remark-sign_remark = text-dca.
INSERT lw_sign_remark INTO TABLE lt_sign_remark.
* Fill the meta data structure
ls_meta-doc_id = p_doc_id.

* Check if a signature instance exists for the meta data
TRY.
    lt_sign = lo_runtime->get_by_meta(
            im_appl         = co_ds_application
            im_object       = co_ds_signobject
            im_meta         = ls_meta ).
  CATCH cx_ds_metadata INTO lo_cx_metadata.

    IF lo_cx_metadata IS BOUND.
      IF lo_cx_metadata->textid =
        cx_ds_metadata=>no_instance_for_metadata.
*       Check note 789924 to get more details why the exception
*       cx_ds_metadata=>no_instance_for_metadata is raised
*       Create a new instance
        l_sign_state = co_state_sig_proc_new.
      ELSE.
*       Internal error
        l_error_text = lo_cx_metadata->get_text( ).
        MESSAGE l_error_text TYPE 'E'.
      ENDIF.
    ENDIF.
ENDTRY.

* Signature process was triggered for meta data before
* table lt_sign filled -> get first entry
IF NOT lt_sign IS INITIAL.
  READ TABLE lt_sign INTO lw_sign INDEX 1.
  lo_sign_inst = lw_sign-sign_inst.
* Check the state of the signature process
  TRY.
      l_sign_state = lo_sign_inst->get_state( ).
    CATCH cx_ds_exception INTO lo_cx_ds_exception.
      IF lo_cx_ds_exception IS BOUND.
*       Internal error
        l_error_text = lo_cx_ds_exception->get_text( ).
        MESSAGE l_error_text TYPE 'E'.
      ENDIF.
  ENDTRY.
ENDIF.

* Check if a new instance must be created for the meta data (not
* necessary if signing procedure is already in process)
IF NOT l_sign_state = co_state_sig_proc_open.
* Create new signature instance for meta data
  TRY.
      CALL METHOD lo_runtime->create
        EXPORTING
          im_appl     = co_ds_application
```



```
          im_object   = co_ds_signobject
          im_meta     = ls_meta
          im_type     = p_ds_typ
          im_strategy = p_si_str
        RECEIVING
          result      = lo_sign_inst.
    CATCH cx_ds_metadata INTO lo_cx_ds_exception.       "#EC NO_HANDLER
    CATCH cx_ds_registry INTO lo_cx_ds_exception.       "#EC NO_HANDLER
    CATCH cx_ds_exception INTO lo_cx_ds_exception.      "#EC NO_HANDLER
  ENDTRY.
  IF lo_cx_ds_exception IS BOUND.
*   Internal error
    l_error_text = lo_cx_ds_exception->get_text( ).
    MESSAGE l_error_text TYPE 'E'.
  ENDIF.
ENDIF.

* Concatenate document key and text for signing
CONCATENATE ls_meta-doc_id p_doctxt INTO l_document SEPARATED BY space.

* Sign it
WHILE l_signing_finished = ''.
  TRY.
      CALL METHOD lo_sign_inst->sign
        EXPORTING
          im_signer            = p_signer
          im_doc_txt           = l_document
          im_doctype           = 'TXT'
          im_remarks           = lt_sign_remark
          im_default_remark_id = l_default_remark_id
        IMPORTING
          ex_sel_remark        = ls_selected_remark.
    CATCH cx_ds_escape INTO lo_cx_ds_escape.
      IF lo_cx_ds_escape IS BOUND.
*       User cancelled the individual signature -> don't change the
*       application data and don't update the database tables
        l_change_appl_data = ''.
        l_signing_finished = ''.
        EXIT.
      ENDIF.
    CATCH cx_ds_message INTO lo_cx_ds_message.
*     Transfer messages (as error messages)
      IF lo_cx_ds_message IS BOUND.
        MESSAGE ID lo_cx_ds_message->msgid
        TYPE 'E' NUMBER lo_cx_ds_message->msgno.
      ENDIF.
    CATCH cx_ds_context INTO lo_cx_ds_exception.        "#EC NO_HANDLER
    CATCH cx_ds_exception INTO lo_cx_ds_exception.      "#EC NO_HANDLER
  ENDTRY.
  IF lo_cx_ds_exception IS BOUND.
*   Internal error
    l_error_text = lo_cx_ds_exception->get_text( ).
    MESSAGE l_error_text TYPE 'E'.
  ENDIF.
*   The application could use the information which remark was
*   selected by the user (ls_selected_remark) to trigger application-
*   specific actions..

* Get signature state
  CLEAR l_sign_state.
  TRY.
      l_sign_state = lo_sign_inst->get_state( ).
    CATCH cx_ds_exception INTO lo_cx_ds_exception.
      IF lo_cx_ds_exception IS BOUND.
*       Internal error
        l_error_text = lo_cx_ds_exception->get_text( ).
        MESSAGE l_error_text TYPE 'E'.
      ENDIF.
  ENDTRY.

* Trigger status changes depending on the status of the
* signature process
  CASE l_sign_state.

    WHEN co_state_sig_proc_closed.
*     Signature process finished
      WRITE:/ text-spf, p_doc_id.
      l_signing_finished = 'X'.
    WHEN co_state_sig_proc_open.
      WRITE:/ text-spo.

    WHEN co_state_sig_proc_canc.
```



```
      WRITE:/ text-spc.
      l_signing_finished = 'X'.
*     Execute application action -> don't set the status, but update the
*     digital signature process
      l_change_appl_data = ''.
    WHEN OTHERS.
*     ERROR!!!
      l_signing_finished = 'X'.
      l_error_ds = 'X'.
  ENDCASE.

* For asynchronous signature processes the WHILE loop must
* be left after one signer signed
  IF p_ds_typ = lo_runtime->co_type_async OR
     p_ds_typ = lo_runtime->co_type_async_ch.
    l_signing_finished = 'X'.
  ENDIF.

  IF NOT ls_selected_remark IS INITIAL.
*   User selected a remark
    WRITE:/ text-srm, ls_selected_remark-sign_remark.
  ENDIF.

ENDWHILE.

CHECK l_error_ds IS INITIAL.
* Update task only if parameter value is set
IF p_savedb = 'X' AND NOT l_change_appl_data IS INITIAL.
  CASE l_change_appl_data.
    WHEN 'I'.
      INSERT dsig_booking_ex FROM ls_dsig_booking_new.
    WHEN 'U'.
      MODIFY dsig_booking_ex FROM ls_dsig_booking_new.
  ENDCASE.
  COMMIT WORK.
  WRITE:/ text-dbu.
ENDIF.

CLEAR: l_error_ds, l_signing_finished.
```


Copyright

