

# APIs for Logistics



HELP.LOAPI

**Release 4.6B**



## Copyright

© Copyright 2000 SAP AG. All rights reserved.

No part of this brochure may be reproduced or transmitted in any form or for any purpose without the express permission of SAP AG. The information contained herein may be changed without prior notice.

Some software products marketed by SAP AG and its distributors contain proprietary software components of other software vendors.

Microsoft®, WINDOWS®, NT®, EXCEL®, Word® and SQL Server® are registered trademarks of Microsoft Corporation.

IBM®, DB2®, OS/2®, DB2/6000®, Parallel Sysplex®, MVS/ESA®, RS/6000®, AIX®, S/390®, AS/400®, OS/390®, and OS/400® are registered trademarks of IBM Corporation.

ORACLE® is a registered trademark of ORACLE Corporation, California, USA.

INFORMIX®-OnLine for SAP and Informix® Dynamic Server™ are registered trademarks of Informix Software Incorporated.

UNIX®, X/Open®, OSF/1®, and Motif® are registered trademarks of The Open Group.

HTML, DHTML, XML, XHTML are trademarks or registered trademarks of W3C®, World Wide Web Consortium, Laboratory for Computer Science NE43-358, Massachusetts Institute of Technology, 545 Technology Square, Cambridge, MA 02139.

JAVA® is a registered trademark of Sun Microsystems, Inc., 901 San Antonio Road, Palo Alto, CA 94303 USA.

JAVASCRIPT® is a registered trademark of Sun Microsystems, Inc., used under license for technology invented and implemented by Netscape.

SAP, SAP Logo, mySAP.com, mySAP.com Marketplace, mySAP.com Workplace, mySAP.com Business Scenarios, mySAP.com Application Hosting, WebFlow, R/2, R/3, RIVA, ABAP, SAP Business Workflow, SAP EarlyWatch, SAP ArchiveLink, BAPI, SAPPHIRE, Management Cockpit, SEM, are trademarks or registered trademarks of SAP AG in Germany and in several other countries all over the world. All other products mentioned are trademarks or registered trademarks of their respective companies.

## Icons

Icon	Meaning
	Caution
	Example
	Note
	Recommendation
	Syntax
	Tip

## Contents

<b>APIs for Logistics .....</b>	<b>5</b>
Working with APIs .....	6
Possible Scenarios .....	7
List of Available Function Groups .....	10
List of Available APIs .....	13
General Processing Rules .....	20
System Messages .....	22
Naming Conventions for APIs .....	23
Identifying Individual Objects .....	24
API Objects .....	25
API Activities .....	26
Assignment of Dependencies .....	28
Generating ABAP Code .....	31
General APIs (CALO) .....	33
Initializing APIs .....	34
APIs for the Log .....	36

## APIs for Logistics

### Purpose

Application Programming Interfaces (APIs) are program interfaces to applications in the SAP System. You can use APIs to transfer data between an external system and the SAP System. You can transfer data between applications in the SAP System without having to use dialog.

### Features

APIs of the following areas are dealt with:

CA (CACL) Classification

CA (CACLCHR) Characteristics

LO (LOVC) Variant configuration

PP Bills of material (BOMs)

LO (ECH) Change engineering management

## Working with APIs

### Features

APIs are designed to be easy to use. This means:

- Simple and easy-to-follow data transfer and consistent interfaces
- Consistent API structure
- Easy access to APIs using specific function groups
- Clearly defined and delimited functionality
- Function macros for specific areas

APIs are used on different levels. The APIs for many dialog functions are defined across different levels. For example:

- Read-only access to (all) data of an object
- Inserting data (such as characteristics and characteristic values) in existing structures (for example, classes and tables)
- Creating new structures, such as change master records, characteristics, classes, configuration profiles, BOMs, and object dependencies
- Classifying objects
- Finding objects via classes
- Configuration
- Access to basic (Customizing) data (not yet supported)

APIs are designed to fulfill the requirements of both internal and external interfaces. APIs are intended to be accessed by both ABAP routines and external programs (Remote Function Call). For this reason, the APIs are based on ABAP Repository structures (RFC compatible).



For more detailed information on using individual APIs, see the function module documentation in the SAP System.

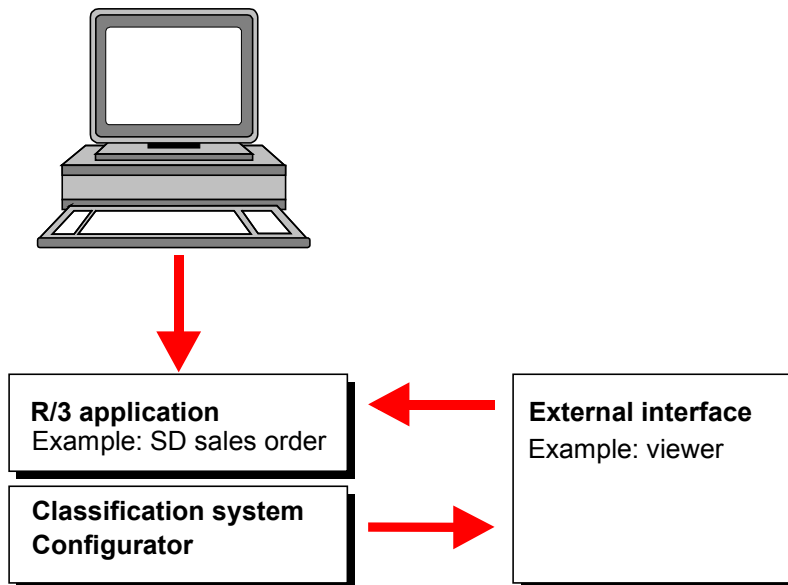
## Possible Scenarios

Application Programming Interfaces (APIs) can be used to support different scenarios. The main distinction is whether the leading system is an SAP System or an external system.

### SAP System as the Leading System

1. The external system is activated from the SAP application.

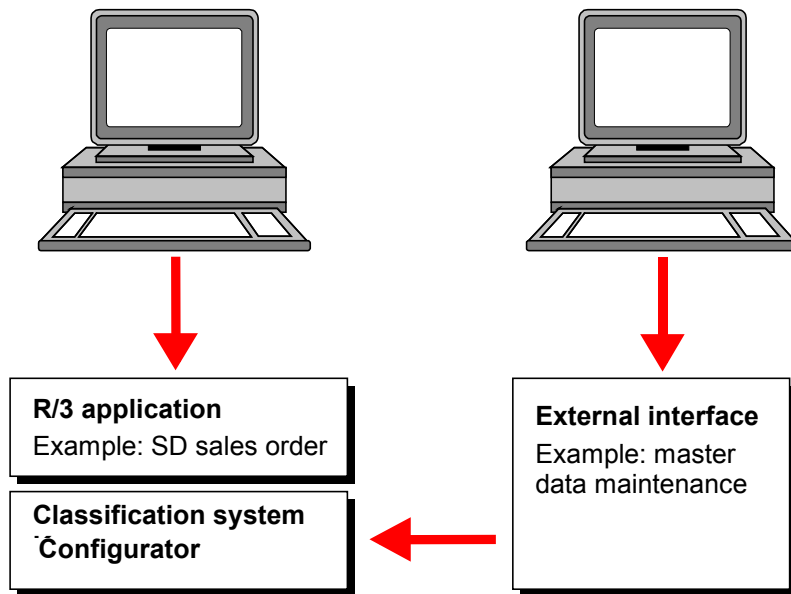
This can be the case if you want to display the result of a configuration in graphical form. APIs are used to transfer the configuration data to the external system. See the following graphic.



2. Data is imported using an API.

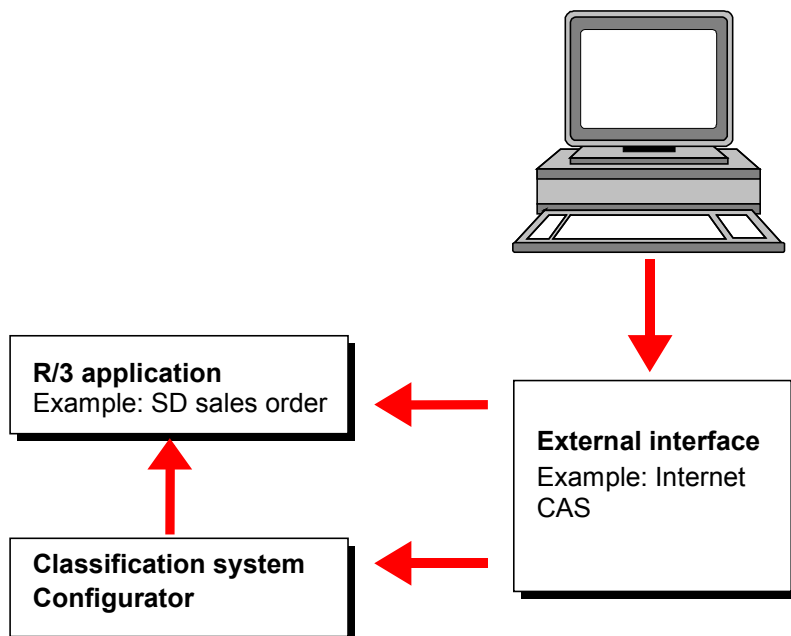
Master data is maintained in an external system. This data is transferred to the SAP classification system or Configurator using APIs at regular intervals. See the following graphic.

**Possible Scenarios**



**External System as the Leading System**

If the external system is the leading system, you can use APIs to access SAP master data and SAP functions, such as the classification system or Configurator. The external system runs as a standalone application or interface, as shown in the graphic below. See the following graphic.



Examples:

**Possible Scenarios**

- From an external sales system that offers products on the Internet, you can use the SAP Configurator to configure a product.
- From a CAD system, you can create characteristics and classes, and classify materials, in the SAP System.

---

**List of Available Function Groups**

## List of Available Function Groups

APIs are grouped together according to their function. We distinguish between:

- General APIs
- APIs for Variant Configuration and Classification
- APIs for BOMs and Engineering Change Management

### General APIs

**Group: Log**

Function group CALO

(APIs for logging)

The function modules in this group are cross-application APIs for logging API calls. These APIs allow you to define parameters for the logging process, read the log, and save the data you require to the log.

### APIs for Variant Configuration and Classification

The APIs for variant configuration and classification are in development class CL. The function groups in this application area are as follows:

**Group: Read Variant Configuration Data**

Function group CARD

(Configurator APIs for Reading Data)

The APIs in this group give you direct read-only access to classification and variant configuration data such as characteristics, classes, objects, object dependencies, and tables.

**Group: Read Characteristics Data**

Function group CACTR

The APIs in this group allow you to read characteristic data, for example, indicators, values, and assigned object dependencies.

**Group: Read Class Data**

Function group CACLR

The APIs in this group allow you to read class data, for example, characteristics of a class, and characteristics, or their values, that have been overwritten.

**Group: Find Objects**

Function group CASE

(Finding objects via classes)

The APIs in this group allow you to find classified objects.  
(set for future development)

**Group: Classification**

Function group CACL

---

**List of Available Function Groups**

(Classification)

The APIs in this group allow you to create new object assignments and value assignment records, and change existing object assignments and value assignment records.



Function group CACL cannot be called in the “Object dependencies” area.

**Group: Maintain Variant Configuration Data**

Function group CAMA

(Maintenance)

The APIs in this group allow you to create and change master data for variant configuration such as configuration profiles, tables, and object dependencies.

**Group: Maintain Characteristics Data**

Function group CACTM

The APIs in this group allow you to maintain characteristic data, for example, creating, deleting, changing, and assigning object dependencies.

**Group: Maintain Class Data**

Function group CACLM

The APIs in this group allow you to maintain class data, for example, creating, changing, deleting, and assigning object dependencies to characteristics and characteristic values.



The APIs in the following CAVC\_ function groups replace the APIs of function group CACO.

**Group: Initialize and Close Configuration**

Function group CAVC\_OBJECTS

The APIs in this group allow you to initialize and close variant configuration. In other words, they process the objects for which variant configuration can be used (such as sales order items).

**Group: Handle Complete Configuration**

Function group CAVC\_CFG

The APIs in this group handle the configuration as a whole (for example, BOM explosion or list of all instances).

**Group: Handle Individual Instances**

Function group CAVC\_INST

The APIs in this group only handle one instance. All APIs in this group have the instance number as an input parameter.

**Group: Dialog Functions**

Function group CAVC\_DIALOG

The APIs in this group use the SAPGUI to process dialog.

---

**List of Available Function Groups****APIs for BOMs and Engineering Change Management**

These APIs are in development classes CS (BOMs) and CC (engineering change management). The function groups in this application area are as follows:

**Group: Read and Maintain Change Master Record Data**

Function group CCAP

The APIs in this group allow you to read, change, and create change master records.

**Group: Read and Maintain BOM Data**

Function group CSAP

The APIs in this group allow you to read, change, and create BOM data.

## List of Available APIs

### CALO

**Group: Log**

CALO_COLLECT_MESSAGES_ON_OFF	Messages are collected internally
CALO_DB_LOG_ON_OFF	Database log on/off
CALO_INIT_API	Initialize API environment
CALO_LOG_INIT_MEMORY	Initialize log interface
CALO_LOG_READ_MESSAGES	Read messages in current log
CALO_MSG_APPEND_DB_LOG	Enter collected messages in API log

### CACL

**Group: Classification**

CACL_CLASSIFICATION_SAVE	Classification: save module
CACL_CLASS_ALLOCATION_MAINT	Classification: maintain assignments of classes to classes
CACL_CLASS_READ_ALLOCATIONS	Classification: read assignments to class
CACL_CLASS_READ_ALLOCATIONS_TD	Classification: read assignments to class TD
CACL_CLASS_READ_VALIDATION	Classification: read assigned values for class
CACL_CLASS_VALIDATION_MAINT	Classification: maintain assigned values for class
CACL_OBJECT_ALLOCATION_MAINT	Classification: maintain assignments for object
CACL_OBJECT_READ_ALLOCATIONS	Classification: read assignments for object
CACL_OBJECT_READ_VALIDATION	Classification: read assigned values for object
CACL_OBJECT_VALIDATION_MAINT	Classification: maintain assigned values for object



Function group CACL cannot be called in the "Object dependencies" area.

### CARD

**Group: Read Variant Configuration Data**

CARD_CNET_CONSTRAINT_READ	Read constraint in constraint net
CARD_CONSTRAINT_NET_READ	Read constraint net
CARD_CON_PROFILE_READ	Read configuration profile

**List of Available APIs**

CARD_DEPENDENCY_READ	Read global dependency
CARD_FUNCTION_READ	Read variant function
CARD_TABLE_READ_ENTRIES	Read variant table entries
CARD_TABLE_READ_STRUCTURE	Read variant table structure
CARD_TABLE_SELECT_LINES	Find variant table lines with specific characteristic values

**CACTR****Group: Read Characteristics Data**

CARD_CHARACTERISTIC_READ	Read attributes and allowed values of characteristic
CARD_CHAR_READ_ALLOC	Read list of dependencies for characteristic
CARD_CHAR_READ_DEP	Read dependencies for characteristic
CARD_CHAR_VAL_READ_ALLOC	Read list of dependencies for characteristic value
CARD_CHAR_VAL_READ_DEP	Read dependencies for characteristic value

**CACLR****Group: Read Class Data**

CARD_CLASS_CHAR_READ_ALLOC	Read list of dependencies for overwritten characteristic
CARD_CLASS_CHAR_READ_DEP	Read dependency for overwritten characteristic
CARD_CLASS_CHAR_VAL_READ_ALLOC	Read list of dependencies for overwritten characteristic value
CARD_CLASS_CHAR_VAL_READ_DEP	Read dependency for overwritten characteristic value
CARD_CLASS_READ	Read characteristics and attributes of class
CARD_CLASS_READ_ALLOC	Read list of dependencies for class
CARD_CLASS_READ_ATTR	Read attributes of class (without characteristics)
CARD_CLASS_READ_CHARACTS	Read characteristics of class (with inherited characteristics and attributes)
CARD_CLASS_READ_DEP	Read dependency for class

**CAMA****Group: Maintain Variant Configuration Data**

List of Available APIs

CAMA_CNET_CONSTRAINT_MAINTAIN	Maintain constraint in net
CAMA_CONSTRAINT_NET_MAINTAIN	Maintain constraint net
CAMA_CON_PROFILE_MAINTAIN	Maintain configuration profile
CAMA_DEPENDENCY_MAINTAIN	Maintain global dependency
CAMA_FUNCTION_MAINTAIN	Maintain variant function
CAMA_TABLE_MAINTAIN_ENTRIES	Maintain variant table entries
CAMA_TABLE_MAINTAIN_ENTRI_LINE	Maintain variant table lines
CAMA_TABLE_MAINTAIN_STRUCTURE	Maintain variant table structure
CAMA_TABLE_MAINTAIN_VALUES	Maintain variant table entries (without dependencies)
CAMA_TABLE_SAVE	Save all data in variant table

**CACTM**

**Group: Maintain Characteristics Data**

CAMA_CHARACT_MAINTAIN	Maintain dependency (create, change, delete)
CAMA_CHARACT_SAVE	Save all data in characteristic
CAMA_CHAR_ALLOCATE_GLOB_DEP	Maintain global dependency for characteristic
CAMA_CHAR_DEL_DEP	Delete dependency for characteristic
CAMA_CHAR_MAINTAIN_DEP	Maintain local dependency for characteristic
CAMA_CHAR_SNGL_MAINTAIN	Maintain single dependency (create, change, delete)
CAMA_CHAR_VAL_ALLOCAT_GLOB_DEP	Maintain global dependency for characteristic value
CAMA_CHAR_VAL_DEL_DEP	Delete dependency for characteristic value
CAMA_CHAR_VAL_MAINTAIN_DEP	Maintain local dependency for characteristic value

**CACLM**

**Group: Maintain Class Data**

CAMA_CLASS_ALLOCAT_GLOB_DEP	Assign global dependency to class
CAMA_CLASS_CHAR_ALLOC_GLOB_DEP	Assign global dependency to characteristic of class
CAMA_CLASS_CHAR_DEL_DEP	Delete dependency for overwritten characteristic

**List of Available APIs**

CAMA_CLASS_CHAR_MAINTAIN_DEP	Maintain local dependency for characteristic of class
CAMA_CLASS_CHVAL_ALL_GLOB_DEP	Assign global dependency to characteristic value of class
CAMA_CLASS_CHVAL_DEL_DEP	Delete dependency for overwritten characteristic
CAMA_CLASS_CHVAL_MAINTAIN_DEP	Maintain local dependency for characteristic value of class
CAMA_CLASS_DEL_DEP	Delete dependency for class
CAMA_CLASS_MAINTAIN	Maintain class (create, change, delete)
CAMA_CLASS_MAINTAIN_DEP	Maintain local dependencies for class
CAMA_CLASS_SAVE	Save all data in class
CAMA_CLASS_SNGL_MAINTAIN	Maintain class (create, change, delete)
CAMA_WWS_CLASS_MAINTAIN	Maintain several classes at once (create, change, delete)



The APIs in the following CAVC\_ function groups replace the APIs of function group CACO.

**CAVC\_OBJECTS**

**Group: Initialize and Close Configuration**

CAVC_O_ORDER_BOM_CANCEL	End processing of order BOM
CAVC_O_ORDER_BOM_INIT	Initialize order BOM
CAVC_O_ORDER_BOM_SAVE	Save order BOM

**CAVC\_CFG**

**Group: Handle Complete Configuration**

CAVC_C_CHECK	Check configuration
CAVC_C_EXECUTE	Execute configuration
CAVC_C_GET_BOM_ITEM_DATA	Returns item data for all instances
CAVC_C_GET_INSTANCES	All instances in configuration
CAVC_C_SET_LEVEL_BOM_EXPLOSION	Define level of detail for BOM explosion
CAVC_C_SET_MODE_FIXING	Define mode for BOM fix
CAVC_C_SET_MODE_INSTANTIATION	Define mode for instantiation

## CAVC\_INST

### Group: Handle Individual Instances

CAVC_I_CHANGE_BOM_ITEM_DATA	Change existing instance (BOM item)
CAVC_I_CHARS_DEL_VALUES	Delete assigned values
CAVC_I_CHARS_DEL_VALUES_USER	Delete all assigned values with user justification
CAVC_I_CHARS_GET_ALLOWED_VALS	Read allowed values for characteristics
CAVC_I_CHARS_GET_ASSIGNED_VALS	Read currently assigned values
CAVC_I_CHARS_GET_ATTRIBUTES	Read characteristic attributes
CAVC_I_CHARS_GET_DEFAULT_VALS	Read default values for characteristics
CAVC_I_CHARS_GET_VALID_VALS	Read currently valid values for characteristics
CAVC_I_CHARS_SET_VALUES	Assign values to characteristics
CAVC_I_CHECK	Check instance
CAVC_I_DELETE_BOM_ITEM	Delete instance (BOM item)
CAVC_I_DUPLICATE_BOM_ITEM	Duplicate instance (BOM item)
CAVC_I_FIX_BOM_ITEM	Fix BOM according to mode set
CAVC_I_GET_BOM_HEADER_DATA	Read header data for instance (BOM header)
CAVC_I_GET_BOM_ITEM_DATA	Read data for instance (BOM item)
CAVC_I_GET_BOM_ITEM_DATA_LIST	Read item data for several instances (BOM items)
CAVC_I_GET_CHARS	Read characteristics of instance
CAVC_I_GET_CLASS_INFO	Read information on class node (specialization, object list, basic data)
CAVC_I_GET_PARENT	Return parent instance of instance
CAVC_I_GET_PROFILE	Read current configuration profile for instance
CAVC_I_GET_STATUS	Read status of instance
CAVC_I_INCONSISTENCIES	Read inconsistency messages for instance
CAVC_I_INSERT_BOM_ITEM	Insert new instance (BOM item)
CAVC_I_INSTANTIATE_BOM_ITEM	Instantiate according to set mode
CAVC_I_SELECT_CHILDS	Return all direct instances of instance
CAVC_I_SET_FIELDS_INST	Fill data fields for instantiation
CAVC_I_SET_STATUS	Set status of instance

## CAVC\_DIALOG

### Group: Dialog Functions

## List of Available APIs

CAVC_D_STRUCTURE_WINDOW	Assign values to configuration structure (dialog)
-------------------------	---

## CCAP

## Group: Read and Maintain Change Master Record Data

CCAP_ALT_DATE_CHANGE	API for engineering change management: change alternative date
CCAP_ALT_DATE_CREATE	API for engineering change management: create alternative date
CCAP_ALT_DATE_DELETE	API for engineering change management: delete alternative date
CCAP_ASSIGN_OBJECT_TO_ALT_DATE	API for engineering change management: assign object to alternative date
CCAP_ECN_CREATE	API for engineering change management: create change master record (complete)
CCAP_ECN_HEADER_CHANGE	API for engineering change management: Change change header and object type data
CCAP_ECN_HEADER_CREATE	API for engineering change management: Create change header and object type data
CCAP_ECN_HEADER_DELETE	API for engineering change management: Delete change number
CCAP_ECN_HEADER_READ	API for engineering change management: check whether change number exists
CCAP_ECN_HEADER_READ_MULTIPLE	API for engineering change management: check whether several change numbers exists
CCAP_ECN_MAINTAIN	API for engineering change management: maintain change master record (complete)
CCAP_REV_LEVEL_MAINTAIN	API for engineering change management: create revision level

## CSAP

## Group: Read and Maintain BOM Data

CSAP_BOM_ITEM_MAINTAIN	API for BOMs: process BOM item
CSAP_DOC_BOM_CREATE	API for BOMs: create document structure
CSAP_DOC_BOM_MAINTAIN	API for BOMs: maintain document structure
CSAP_DOC_BOM_READ	API for BOMs: display document structure
CSAP_MAT_BOM_ALLOC_CREATE	API for BOMs: create material BOM

List of Available APIs

CSAP_MAT_BOM_CLOSE	API for BOMs: close/save material BOM after changes
CSAP_MAT_BOM_CREATE	API for BOMs: create material BOM
CSAP_MAT_BOM_DELETE	API for BOMs: delete material BOM
CSAP_MAT_BOM_ITEM_SELECT	API for BOMs: select items of material BOM
CSAP_MAT_BOM_MAINTAIN	API for BOMs: maintain material BOM
CSAP_MAT_BOM_OPEN	API for BOMs: open material BOM for changes
CSAP_MAT_BOM_READ	API for BOMs: display material BOM
CSAP_MAT_BOM_SELECT	API for BOMs: select material BOM(s)

## General Processing Rules

## General Processing Rules

There are API modules for maintaining all objects, such as characteristics, classes, and object dependencies. These APIs allow you to create, change, and delete the object.

Before you can use APIs to maintain objects, specific data must be transferred to the API, either as individual parameters or in internal tables.

### Example: Creating a Characteristic

You want to create the characteristics COLOR and LENGTH, with the description in English "Paintwork" and "Tube length" and characteristic status "2". To do this, transfer the following information to internal table *Characteristic attributes*:

Characteristic name	=	'COLOR'
Status	=	'2'
Data type	=	'CHAR'
Number of characters	=	'9'

Characteristic name	=	'LENGTH'
Status	=	'2'
Data type	=	'NUM'
Number of characters	=	'4'
Negative values allowed	=	'X'

In addition, transfer the following data to internal table *Descriptions*:

Characteristic name	=	'COLOR'
Language	=	'E'
Description	=	'Paintwork'

Characteristic name	=	'LENGTH'
Language	=	'E'
Description	=	'Tube length'

General Processing Rules

With this information, you can call and execute function module CAMA\_CHARACTER\_MAINTAIN. The characteristic is then created in the SAP System.

**Example: Changing a Characteristic**

You want to change the existing characteristic COLOR. You want to change the characteristic status from "2" to "1" and change the English description from "Paintwork" to "Paintwork color".

To do this, call function module CAMA\_CHARACTER\_MAINTAIN with the following input for internal table *Characteristic attributes*:

Characteristic name	=	'COLOR'
Status	=	'1'

In addition, transfer the following data to internal table *Descriptions*:

Characteristic name	=	'COLOR'
Language	=	'E'
Description	=	'Paintwork'
Deletion indicator	=	'X'

Characteristic name	=	'COLOR'
Language	=	'E'
Description	=	'Paintwork color'
Deletion indicator	=	SPACE

First the old description is deleted. Then the new description is inserted.



1. If you try to change an object that does not yet exist, the object is created.
2. If you try to delete an object that does not yet exist, your entry is ignored.

## System Messages

# System Messages

## Exceptions

Exceptions are used for all errors and are always listed at the end of an API, so that different errors can be found and logged.

The API checks all entries. It produces a log of any errors that occur, and sets an ERROR exception for each error.



Under certain circumstances, an exception may be set before the API has checked all entries.

Usually, the only exceptions set are ERROR exceptions. This means that processing was terminated – because essential data was missing, for example.

Some function modules also use a WARNING exception or return code. In this case, a warning message is written to the log.

For each error found, an entry is also written to the log.

## Log

The messages that would normally appear in dialog mode are written to a database log. All types of message (error, warning, and success messages) are saved. This explains all process flows, even in case of complex errors.

## API for Reading the Message Log

There is an API module especially for reading the log before it is saved to the database. The messages and some message attributes are transferred to an internal table. After each API that is called externally, the collected messages (in mixed form) are available to this API. You can evaluate the message identifiers internally in your program, and present the messages in your user interface.

See also [General APIs \[Page 33\]](#)

## Naming Conventions for APIs

All the names used are informative and in English. This is especially important for field and structure names. Only CHAR and NUMC fields are used in APIs.

The following conventions are used for naming APIs:

xxxx\_Object\_Action\_Result

The name can comprise the following elements:

{-}•	xxxx	Function group
{-}•	Object	Object / key (What is it?)
		(see <a href="#">API Objects [Page 25]</a> )
{-}•	Activity	Activity to be performed, for example: read
		(see <a href="#">API Activities [Page 26]</a> )
{-}•	Result	The expected result, for example: characteristics of a class



CAMA\_CLASS\_MAINTAIN  
CSAP\_MAT\_BOM\_CREATE

---

**Identifying Individual Objects**

## Identifying Individual Objects

Identifying objects, that is the object key, is very important within classification. You must provide the object type and object key for the functional module as to which objects you want to classify. The import parameter OBJECT\_TYPE indicates the object table. You can, for example enter the MARA for materials as an object table.

You can derive the key fields for the import parameter in OBJECT\_TYPE Customizing *Classification* These key fields identify the individual objects. Choose the area *Classes* and the IMG action *Maintain object key*. You can call the key fields for the table by double clicking on the table name.

An example of how an object table for a document could look follows:

<u>Field</u>	<u>Value</u>
DOKAR	'DRAW'
DOKNR	'123'
DOKVR	'2'
DOKTL	'1'

## API Objects

The following names can be used in the name of a function module to refer to objects:

BOM	Bill of material
CH_VALUE, VALUE	Characteristic value
CHARACT(ERISTIC)	Characteristic
CLASS	Class
CON(FIGURATOR)	Variant Configurator
CON_RESULT	Configuration result
DB_LOG	Message log in database
ECN_HEADER	Change master header
MAT_BOM	Assignment of material to BOM
MATERIAL	Material number
MSG	Message from API to calling program
OBJ_TYPE	Object type
OBJECT	SAP object (for example, material)
SEL_RESULT	Search result in classification system
INSTANCE	Instance as identifier of object in configuration structure
(VARIANT-) TABLE	Variant table

---

**API Activities**

## API Activities

The following names can be used in the name of a function module to refer to activities:

- **Create**  
Create a new (API) object.
- **Insert**  
Insert, which also means creating, but the position in a list is important – for example, a characteristic value in an existing list of values.
- **Append**  
Add to the end of a list.
- **Read**  
Read object data – for example, read class attributes.
- **Check**  
Check
- **Change**  
A change is usually made by deleting existing data and writing new data. The writing is done with the "maintain" module.
- **Init**  
Initialize
- **Prepare**  
Prepare
- **Fix**  
Fix
- **Define**  
Define
- **Execute**  
Execute
- **Activate**  
Activate
- **Deactivate**  
Deactivate
- **Convert**  
Convert
- **Delete**  
Delete

- **Select**  
Select or find

## Assignment of Dependencies

## Assignment of Dependencies

### Use

You can use APIs to assign object dependencies to the following objects, as in dialog mode:

- Characteristic
- Characteristic value
- BOM item
- Configuration profile

There are 6 types of dependency:

1	CON	Constraint
2	PRE	Precondition
3	ACT	Activity
4	SEL	Selection condition
5	CNET	Constraint net
6	PROC	Procedure

You have the option of entering either the number or the key for the dependency type. The read modules always display the external keys.

There are 2 methods for maintaining and reading dependencies for the following function groups:

- CSAP
- CARD and CAMA

### Rules for Modules in Group CSAP



This method is also used to assign dependencies in modules CAMA\_CON\_PROFILE\_MAINTAIN (maintain configuration profile) and CAMA\_TABLE\_MAINTAIN\_ENTRIES (maintain table entries).

There are maintenance modules for the individual objects, containing 5 dependency data tables in the interface. These tables allow you to assign local and global object dependencies to BOM items.

- A local dependency does not have an internal name until you save it, so you enter an external name when you create a local dependency, and this name is used to refer to the various tables containing data that belongs to the dependency.
- If you want to assign global dependencies, you enter the name of the dependency in the *Internal dependency name* field. You use table DEP\_ORDER to assign global dependencies.

The tables contain the following data:

- Table DEP\_DATA (object dependencies basic data)

**Assignment of Dependencies**

This table is for transferring the basic data of a dependency.

- Internal dependency name
- External dependency name
- Dependency type
- Status
- Dependency group
- Authorization profile
- Deletion indicator

- Table DEP\_DESCR (object dependencies descriptions)

This table is for transferring the language-dependent descriptions of a dependency.

- Internal dependency name
- External dependency name
- Language
- Description
- Deletion indicator

- Table DEP\_ORDER (assigned dependencies)

- Internal dependency name
- External dependency name
- Counter for sort sequence of objects

This table shows the dependencies that are assigned to an object. To delete an assignment, enter a deletion indicator in this table or table DEP\_DATA. This deletes the assignment for a global dependency, but deletes a local dependency completely. This deletes the assignment for a global dependency, but deletes a local dependency completely.

- Table DEP\_SOURCE (object dependencies source code)

This table is for transferring the source code of a dependency.

- Internal dependency name
- External dependency name
- Line number
- Line for source code

- Table DEP\_DOC (object dependencies document)

- Internal dependency name
- External dependency name
- Language
- Line number

### Assignment of Dependencies

- Text format

You can enter long texts for dependencies. In field TXT\_FOR in table DEP\_DOC, you can enter data in SAPscript format, for example:

- AS Standard paragraph
- B1 First-level list
- N1 Numbered list

## Rules for Modules in Groups CARD and CAMA

To assign local dependencies, your data must be complete, because a local dependency does not have a name that that application programmer can see, unlike global dependencies.

Each new call creates a new dependency. If you set the deletion indicator, you delete all local dependencies. You can create actions, procedures, selection conditions, and preconditions as local dependencies. You cannot create constraints as local dependencies.

You must enter the following data when you create local dependencies:

- Language
- Description
- Status
- Dependency type
- At least one line of source code

A log shows the cause of any errors.

To assign a global dependency, you use the name that was assigned when the dependency was created. This name remains during the effectivity period of the dependency and is unique. The dependency is assigned by saving its name in an internal table. You can create or delete individual global dependencies.

When dependencies are read, all local dependencies are provided with all available data (such as description and source code). Each call returns a dependency. Global dependencies appear as a list of assignments in a separate table.

The structure of the tables matches the method for CSAP modules. However, there are no name fields, and the basic data of a dependency is transferred as an individual structure, not as a table.

## Generating ABAP Code

You can generate your own ABAP code from the API function modules using transaction GENC. To do this, proceed as follows:

1. Enter the name of your program on the initial screen. Confirm your entry.
2. You see the function structure of your program on the next screen.

This structure always contains function module CALO\_INIT\_API (see [Initializing APIs \[Page 34\]](#)). You cannot delete this function module. The structure contains the description *Initialize Logfile* for this function module.

The start and end of a function module are marked by special character strings. The description of a function module that is to be shown in the structure is entered after the string that marks the start of a function module. You can change this description if required.

**\*&BPAPILogfile&:Initialize Logfile**

perform api\_init\_logfile.

**\*&EPAPILogfile&**

You can assign additional API modules that are predefined by SAP to the program. To do this, choose *Edit → New functions*. You see a dialog box showing a list of APIs according to the areas they belong to. To copy a function module across, double-click on it.

If a function module contains optional data, you can choose which parameters you want transferred to the interface.

For example, in characteristics maintenance functions, you can define whether the basic data, characteristic description, values, or value descriptions are transferred to the interface as parameters.

You can also define whether you want a characteristic to be created or changed by the module. Finally, you generate the function module in order to copy it to your program.

3. You can use the *Program* function to display or change the source code. For example, you can include your own functions in the program. If you want these to be entered in the function structure, mark the start and end of the function using the following character strings:

*&BPUSR&	Start of function
	After this key, you enter the name that appears in the function structure
*&EPUSR&	End of function

1. The data that the program requires for processing is shown as a place holder (<place holder>). The place holder must be replaced by actual values.

In the *Maintain characteristic* module, for example, if you select basic data and characteristic values as the parameters for the interface, the following place holders must be replaced:

```
* Define basic data
  perform ct_basics tables ct_characts
```

---

**Generating ABAP Code**

```
                using      '<ct_name>'
                    '<ct_type>'
                    <ct_length>
                    '<ct_status>'.

* Define value
perform ct_value tables ct_char_vals
        using
        '<ct_name>'
                    '<ct_value>'
                    '<ct_default>'.

```

2. To define whether log entries are written to the database for each module, or whether all messages are collected and read at the end of the program, choose *Program* → *Log settings*.

If you define the setting *Collect messages*, function module CALO\_MSG\_APPEND\_DB\_LOG must be called in order to write the log entries to the database (see [APIs for the Log \[Page 36\]](#)).

## General APIs (CALO)

General APIs are for general functions, as opposed to the data of a specific application. These APIs are used to initialize the API environment, define the log functions, and access the log.

## Initializing APIs

## Initializing APIs

Function module CALO\_INIT\_API is for initializing the API environment. All transfer parameters have default values.

If you want to use APIs for classes, variants, BOMs, and engineering change management, you **must** call this module first.

INPUT:	<b>FLAG_DB_LOG_ON</b> (Database log on)	Default: X
	<b>FLAG_MSG_ON</b> (Transfer messages to internal table)	Default: X
	<b>FLAG_API_API_CALL_ON</b> (API called by API → collect messages)	Default: Space
	<b>FLAG_COLLECT_MSG_ON</b> Collect messages	Default: Space
	<b>EXTERNAL_LOG_NO</b> External log number (see transaction SLG1)	Default: API
	<b>DEL_LOG_AFTER_DAYS</b> Number of days after which database log is deleted	Default: 10
	<b>DATA_RESET_SIGN</b> deletion indicator (Resets data field to initial value)	Default: Space

If indicator *Database log* or indicator *Transfer messages* is selected, messages produced by APIs are written to local memory.

If indicator *Database log* is selected and indicator *Collect messages* is not selected, each API writes a log entry to the database that contains all messages.

If indicator *Collect messages* is selected, the messages are collected until the application program calls the "Write log" API.

If indicator *Collect messages* is not selected (= SPACE), the local log memory is initialized at the start of each API.

If indicator *Collect messages* is selected, the API user must read the log entries as required and initialize the log interface.

If you have entered a number of days after which the database log is deleted, the log cannot be deleted before this number of days has passed. If you enter the value "-1", you can delete the log immediately.



## APIs for the Log

## APIs for the Log

## Switching the Log on and off: CALO\_DB\_LOG\_ON\_OFF

This function module is used to switch the log process on and off. The default setting for module CALO\_INIT\_API switches the log on (see [Initializing APIs \[Page 34\]](#)).

- If indicator FLAG\_DB\_LOG\_ON is not selected (SPACE), the log can be switched off. If the log is switched off, no messages are entered in the database log.
- If indicator FLAG\_DB\_LOG\_ON is selected (X), the log is switched on. Once the log is switched on, messages are collected internally.



If the log object or sub-object cannot be found, it must be created with transaction SLG0.

Object:	CAPI
Sub-object:	CAPI_LOG

INPUT:

FLAG_DB_LOG_ON	Default: X	(Database log on)
----------------	------------	-------------------

## Write Log Entry to Database: CALO\_MSG\_APPEND\_DB\_LOG

You can use this function module to write the messages collected in local memory to the database log.

The internal memory is then initialized.



Your program calls a large number of APIs, and you do not want all calls to write a log entry to the database. You want the APIs to write the messages to local memory only, and you want to write the collected messages to the database yourself at points in time that you decide.

This is what you do:

- 1) Call function module CALO\_INIT\_API with *Collect messages* switched on.
- 2) Call the APIs you require. All of these APIs write their messages to local memory only.
- 3) Call function module CALO\_MSG\_APPEND\_DB\_LOG when you want to write a log entry (with all messages) to the database and initialize local memory.

## Read Messages in Current Log: CALO\_LOG\_READ\_MESSAGES

You can use this function module to read the contents of the local log memory.

This does not delete the local memory until you do one of the following:

- Write the log to the database
- Call the API for initializing the log interface

INPUT:	Class of message	Default: 4
	Allowed values:	
	1: Read top priority logs only	
	2: Read medium priority logs also	
	3: Read low priority logs also	
	4: Read all logs	
	<b>Language</b>	

### Initialize Log Interface: CALO\_LOG\_INIT\_MEMORY

This function module deletes the local log memory.

### The Log

You can use transaction SLG1 to find data for object CAPI, sub-object CAPI\_LOG. A log entry contains all messages of a "Write log" call.