SAP
Records Management

Document Management in Records Management

Documentation for Developers

March 12, 2004

Contents

1 Introduction

This document describes the architecture of the component Document Management within Records Management, as well as the options available for changing or enhancing the existing functions.

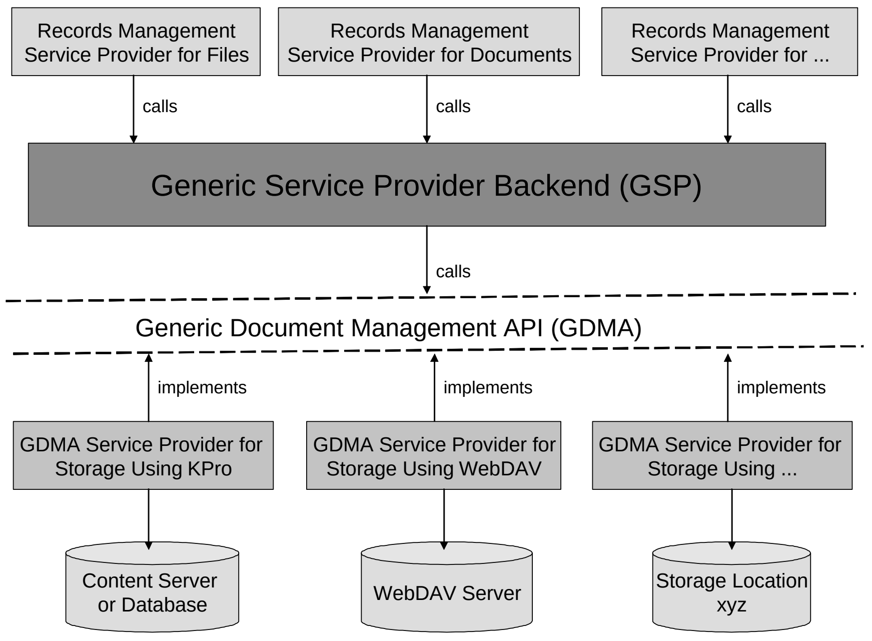
To fully understand this document, you require knowledge of ABAP objects, as well as knowledge of all the technical terms and the architecture of the Records Mana gement Framework. You can find information about this in the Records Management reference documentation for developers.

The Document Management architecture enables Records Management elements to be stored in different repositories. Externally, however, these different repositories are not visible; the user interface remains the same.

The central part of Document Management is the Generic Document Management API (referred to in the following as GDMA). This is defined through a set of interfaces, which are implemented by GDMA Service Providers. A GDMA Service Provider defines where the elements are stored. It is registered in a separate area of the Records Management Framework. In Customizing (registry maintenance), you can configure for each type of Records Management element, which of these GDMA Service Providers (that is, which storage location) is selected. SAP delivers two GDMEA Service Providers: Service Provider for storing using the Knowledge Provider and Service Provider for storing using WebDAV.

The GDMA is called by the Generic Service Provider Backend. In turn, the Generic Service Provider Backend offers an API that is called by Records Management Service Providers. A Records Management Service Provider is responsible for displaying an element with in Records Management. All Records Management Service Providers that generate data and administer in a repository use the Generic Service Provider Backend.

The following diagram gives you an overview of the architecture:


Records Management Service Provider for Files
Records Management Service Provider for Documents
Records Management Service Provider for ...
calls
calls
calls
Generic Service Provider Backend (GSP)
calls
Generic Document Management API (GDMA)
implements
implements
implements
GDMA Service Provider for Storage Using KPro
GDMA Service Provider for Storage Using WebDAV
GDMA Service Provider for Storage Using ...
Content Server or Database
WebDAV Server
Storage Location xyz

Because the architecture is conceived openly, you can implement separate Records Management Service Providers and GDMA Service Providers.

To implement a Records Management Service Provider that is based on the Generic Service

Provider Backend, you require some knowledge about the Generic Service Provider Backend. You can find such documentation in chapter 2. You can find more information about implementing a Records Management Service Provider in the tutorial *Implementing a Service Provider* and also the Records Management reference documentation for developers.

To implement a GDMA Service Provider, you require some knowledge about the GDMA. You can find this documentation in chapter 4.

You can also use the GDMA independently of the Generic Service Provider Backend and independently of Records Management. You can find documentation about the GDMA from a caller's point of view in chapter 3.

2 Generic Service Provider Backend

You can use the Generic Service Provider Backend (short form *Generic Service Provider*, referred to in the following as GSP) if you want to implement a Records Management Service Provider that generates and stores its own data. Among the Service Providers that are delivered by SAP as standard, the SP for files and the SP for documents both use the GSP, for example. In the following, *all* elements that are stored using the GSP are referred to as *documents*.

2.1 The Document Concept

As a document and the attributes of a document must be versionable, the GSP uses the following document concept.

There is an entity for the document as a whole, with all of its existing versions, which is virtually the shell of the document without its content. In the Knowledge Provider (KPro), this entity is called the *Logical Information Object* (LOIO). In the Generic SP Backend, the LOIO is represented by the interface IF_SRM_DOCUMENT.

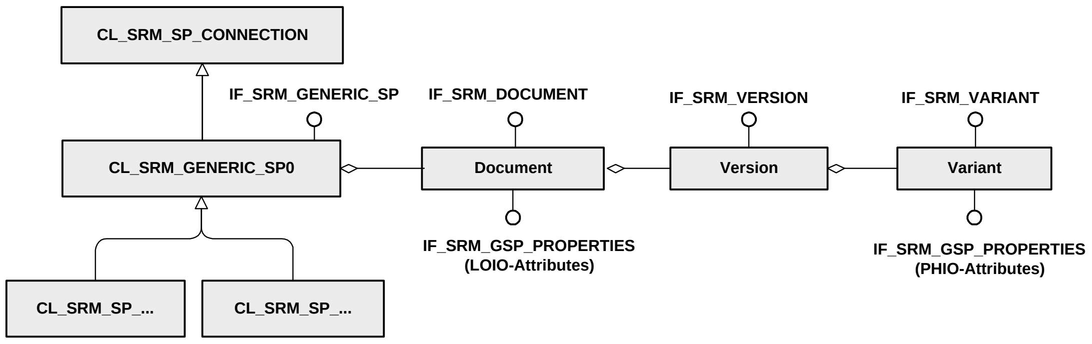
In Customizing, you can set attributes in the Content Model for the LOIO, namely LOIO attributes. These refer to all versions of a document. If a user changes an LOIO attribute value for a version of the document, then this change is also valid for all previous versions of the document.

There are two entities for the versions of a document: *versions* and *variants*. A version is a content version of a document. In terms of its contents, a variant is identical to a version, however it is in a different language or format. The first variant is always the original variant, that is, it is identical to the version. In the KPro, versions and variants are known as *Physical Information Objects* (PHIOs). If you want to access the content of a document, you must access the variants.

In the Generic SP Backend, versions are represented through the interface IF_SRM_VERSION, and variants through the interface IF_SRM_VARIANT. Versions and variants are numbered sequentially. You can always call the current version and the variant that fits the logon language by using the value 0.

In Customizing, you can also set attributes in the Content Model for PHIOs, namely PHIO attributes. PHIO attributes are saved specific to each version. If a user changes a PHIO attribute value for a version of the document, then this change is not made in all previous versions of the document as well.

The following diagram gives you an overview of the connectivity between entities in the Generic SP Backend:



If you want use the Generic SP Backend, you must allow the backend class of your Service Provider to inherit from the Basis class CL_SRM_GENERIC_SP0. This class already fulfils the class roles IS_SP_SYSTEM_PARA and IS_SP_CONTENT_CONNECTION_CLASS.

2.2 Overview of Interface Methods

The inheritance hierarchy provides you with the interface methods that are documented below.

2.2.1 IF_SRM_GENERIC_SP

The inheritance gives you a reference to this interface. The interface IF_SRM_G ENERIC_SP is responsible for the general connection to the store.

Method Name	**Explanation**
CREATE_DOCUMENT	Generates a new document instance for the current POID (model POID) in the repository, and sets the instance POID. The import parameters have the following meanings: PROPERTY: Table of attributes being set. DOCUMENT_ID: Value for attribute *Document ID*. Can only be set if you have not set the document ID in the attribute table. DOC_OBJECT_ID: Technical document ID (GUID, without document class). Must be set if you check the flag DO_UPDATE_TASK (see below). DO_COMMIT: Does a commit have to be performed after the document has been generated? Default: No. DO_UPDATE_TASK: Does the document have to be generated in the update task? Default: No.
GET_DOCUMENT	Returns the document that the SP is using. You receive a reference to IF_SRM_DOCUMENT.
GET_DOC_CLASS	Reads for an element type the document class for the KPro store (corresponds to the value of the connection parameter DOCUMENT_CLASS, the class of the LOIOs).
GET_PROPERTY_TYPES	Reads the attribute properties of a document class. Use parameter PROP_LOCATION to specify whether the properties of the LOIO attributes or PHIO attributes are to be read. As a value, you assign one of the constants srmgs_c_prop_loc_doc (for LOIO attributes) or srmgs_c_prop_loc_ver (for PHIO attributes).

GET_QUERY	Returns reference to the interface for the search IF_SRM_GSP_QUERY.
IS_AUTHORIZED	Checks whether a user has authorization for an activity. In the case of model POID activities, CREATE, SEARCH, and for instance POID activities, VIEW, MODIFY, CLOSE, REOPEN, DELETE. You can find corresponding constants in the interface IF_SRM_DOCUMENT.
COPY_PARTIAL	Copies exactly one version/variant of the document. Returns the POID of the new document. Internally, the method IF_SRM_DOCUMENT~COPY_PARTIAL is called.
CREATE_SP_POID	Creates the SP-specific part of a POID for a document. This method is not usually used, as this is automatically done during CREATE_DOCUMENT for example.

2.2.2 IF_SRM_DOCUMENT

You can receive a reference to this interface through IF_SRM_GENERIC_SP~GET_DOCUMENT( ). The interface IF_SRM_DOCUMENT is responsible for operations on the document as a whole (LOIO).

Method	**Explanation**
APPLY_LOCK	Locks the document together with all its versions and variants for parallel changes out of other sessions.
REMOVE_LOCK	Removes the lock.
NEW_VARIANT	Generates a new version with a new variant. Using the parameter CREATE_NEW_LOG_VERSION, you specify whether the preceding variant is to be overwritten (srmgs $\rightarrow$ false) or is to remain accessible (srmgs $\rightarrow$ true). You can only execute this method if the entire document has been previously locked by calling the method APPLY_LOCK.
GET_VARIANT	Gets a variant of the document. You get a reference to IF_SRM_VARIANT. This method is generally used for accessing the current content of a document. Calling this method has the same effect as calling GET_VERSION and then calling GET_VARIANT for the version. The following rules (in the given order) are used for the context resolution: If version and variant were handed over explicitly, then these are used. If they were not transmitted, then the default value is –1. This has the effect that the values are copied from the current POID. If the version and variant have not been specified in the POID, the current version and the variant that fits the logon language are used.
CREATE_VERSION	Generates a new version of the document. May be better to call NEW_VARIANT immediately.
GET_VERSION	Gets a version of the document. May be better to call GET_VARIANT immediately.

GET_VERSIONS_INFO	Gets attribute values of all versions of the document. You transfer a list containing the attributes for which you want to read the values. Here, the names of the attributes are the determining factor, the column *Value* is not evaluated. If the list is empty, then all attributes of the versions are returned.
DELETE	Deletes the document together with all its versions and variants. You can only execute this method if the entire document was previously locked by calling the method APPLY_LOCK.
GET_PROPERTY_INTERFACE	Returns reference to the interface IF_SRM_GSP_PROPERTIES for accessing the LOIO attributes.
IS_MODIFIABLE	Checks whether the document can be modified or whether it is locked.
IS_AUTHORIZED	Checks whether the user has authorization for one of the instance activities VIEW, MODIFY, CLOSE, REOPEN and DELETE. You can find corresponding constants on the interface IF_SRM_DOCUMENT.
GET_REPOSITORY_TYPE	Reads the repository type. It returns one of the constants srmgs_c_repository_kpro or srmgs_c_repository_webdav. Note: If you use a different storage location, you have to redefine this method.
GET_DOC_ID	Reads the document ID.
COPY_PARTIAL	Copies a given version and variant. Returns the document ID of the new document.
SET_UPDATE_MODE	Sets the update mode. Note: if the data is to be stored on a content server, update mode is not possible.
GET_UPDATE_MODE	Reads the update mode.
SET_COMMIT_MODE	Sets the commit mode.
GET_COMMIT_MODE	Reads the commit mode.
FREEZE_CURRENT_VERSION	Closes the current version, meaning that it can no longer be changed.

2.2.3 IF_SRM_VERSION

You get a reference to this interface through IF_SRM_DOCUMENT~ GET_VERSION( ). The interface IF_SRM_VERSION is respon sible for operations in a version of the document.

Method Name	**Explanation**
CREATE_VARIANT	Generates a new variant for this version.
GET_VARIANT	Reads a particular variant of this version.
GET_VARIANTS_INFO	Returns attribute values for all variants of this version.
GET_COMPONENTS_INFO	Returns information about components of the document.
DELETE	Deletes the version.

2.2.4 IF_SRM_VARIANT

You get a reference to this interface through IF_SRM_DOCUMENT~GET_VARIANT( ) or IF_SRM_VERSION~GET_VARIANT( ). The interfac e IF_SRM_VARIANT is responsible for operations in a variant of the document.

Method Name	**Explanation**
GET_PROPERTY_INTERFACE	Gets the interface IF_SRM_GSP_PROPERTIES for accessing the PHIO attributes.
GET_COMPONENTS_INFO	Returns information about the components of the document.
DELETE	Deletes the variant.

2.2.5 IF_SRM_GSP_PROPERTIES

The interface IF_SRM_GSP_PROPERTIES is responsible for operations in attribute values.

For accessing LOIO attributes, you get the reference to the interface through IF_SRM_DOCUMENT~GET_PROPERTY_INTERFACE( ). For accessing PHIO attributes, you get the reference to the interface through IF_SRM_VARIANT~GET_PROPERTY_INTERFACE( ).

Method Name	**Explanation**
SET_PROPERTIES	Sets the attribute values. You transfer a list of the attribute name / attribute value pairs.
GET_PROPERTIES	Reads the attribute values. In the changing parameter, you transfer a list of attribute names whose values you want to read, and receive the complete list (names and values) in return. If you want to read all attribute values, transfer an empty list. Parameter PROPS_DELETED gives you information as to whether attributes were deleted from the list due to missing read authorization.
DELETE_PROPERTIES	Deletes the attributes that you transfer.
GET_PROPERTY_TYPES	Reads the properties of all attributes.
GET_PROPERTY	Reads an attribute.
GET_MAINT_PROPERTIES	Reads the maintainable attributes of the document.
CLEAR_CACHE	Deletes buffered attribute values.

2.2.6 IF_SRM_GSP_TAB_TRANSFER

You get a reference to this interface through casting the r eference to IF_SRM_VARIANT to IF_SRM_GSP_TAB_TRANSFER. This interface is responsible for transferring content using internal (binary or ASCII) tables.

Method Name	**Explanation**
GET_CONTENT	Reads the content of the variant. You can use the AS_ASCII flag to define whether you want to transfer the content in binary or ASCII format. The default setting is binary. Depending on which flag you set, you receive table ASCII_CONTENT or

	BIN_CONTENT. The table COMPONENTS contains detailed information of the individual components (see below).
SET_CONTENT	Sets the content of the variant. You can only execute this method if the entire document was previously locked by calling the method IF_SRM_DOCUMENT~APPLY_LOCK. If the content is in binary format, you transfer table BIN_CONTENT, and if it is in ASCII format, table ASCII_CONTENT. In both cases you have to transfer table COMPONENTS. The fields mean the following: • comp_count: Listing number, not persistent, meaning that a component can get different values of comp_count when being read. • comp_id: File name (must include the correct extension) • mimetype: MIME type • comp_size: Binary size, number of bytes of content • comp_num: Listing number, stored persistently in the repository. We recommend that you deal with comp_count and comp_num in the same way • binary_flag: SRMGS_TRUE: The content is in binary format • first_line: Number of row where the content of the component starts in the internal table (for the first component, starts at 1, not 0) • last_line: Number of row where the content of the component ends.

2.2.7 IF_SRM_GSP_FILE_TRANSFER

You get a reference to this interface through casting the re ference to IF_SRM_VARIANT to IF_SRM_GSP_FILE_TRANSFER. This interface is responsible for transferring content using files.

Method Name	**Explanation**
GET_CONTENT	Reads the content of the variant.
SET_CONTENT	Sets the content of the variant. You can only execute this method if the entire document was previously locked by calling the method IF_SRM_DOCUMENT~APPLY_LOCK.

2.2.8 IF_SRM_GSP_URL_TRANSFER

You get a reference to this interface through casting the r eference to IF_SRM_VARIANT to IF_SRM_GSP_URL_TRANSFER. This interface is responsible for transferring content using a URL.

Method Name	Explanation
GET_URL_FOR_GET	Gets the URL for reading the content. In the import parameter URL_LIFETIME, you transfer the lifetime of the URL. Two values are available: SRMGS_LIFETIME_VOLATILE (the URL can only be assigned to a data provider once) and SRMGS_LIFETIME_TRANSACTION (the URL is valid for the entire transaction). You use the flag WEB_URL_ONLY to specify whether the URL is to be a data provider URL or an application server URL (Web URL). Note: if you want to use the Office integration, we recommend that you choose the data provider URL, otherwise problems may occur.
GET_URL_FOR_PUT	Gets the URL for writing the content.
CONFIRM_PUT	Copies data (call after GET_URL_FOR_PUT).

2.2.9 IF_SRM_GSP_QUERY

You get a reference to this interface through IF_SRM_GENERIC_SP~GET_QUERY( ). The interface IF_SRM_GSP_QU ERY is responsible for the search. It stands apart from the other interfaces, since it does not deal with operations on a document instance.

Method Name	Explanation
EXECUTE	Performs a search. You transfer the following parameters: QUERY_PARAMS: Table con taining the search requirements entered by the user (attributes, operators, attribute values). The individual rows of the table contain search requirements, all of which must be satisfied by the documents simultaneously, for them to be included in the results list. MAX_HITS: Maximum number of hits to be displayed in the results list. SEARCH_PROPS: Internal use only. REQUEST_PROPS: Attributes to be displayed in the results list. The following parameters are returned: RESULTS: Results list. RESULT_PROPS: Att ributes and attribute values of the individual results.
IS_PROP_VALUE_UNIQUE	Checks whether an attribute/attribute value pair is unique.

2.3 Examples

2.3.1 Generating a Document with Version and Variant

Coding Example

```
  DATA: lif_gsp_query      TYPE REF TO if_srm_gsp_query,
        i_document_id      TYPE string,
        l_is_unique        TYPE srmgs_boolean,
        lt_properties      TYPE srmgs_property_tab,
        wa_properties      TYPE srmgs_property,
        lif_document       TYPE REF TO if_srm_document,
        lif_variant        TYPE REF TO if_srm_variant,
        lif_tab_cont              TYPE REF TO if_srm_gsp_tab_transfer,
        component_tab      TYPE srmgs_components,
        ascii_content      TYPE srmgs_ascii_content,
        ex_gsp             TYPE REF TO cx_srm_gsp.

  TRY.

** Check whether document ID is unique
      lif_gsp_query = me->if_srm_generic_sp~get_query( ).

      CALL METHOD lif_gsp_query->is_prop_value_unique
        EXPORTING
          prop_name   = if_srm_document=>prop_document_id
          prop_value  = i_document_id
          only_actual = 'X'
        RECEIVING
          is_unique   = l_is_unique.

** Set document ID and description for LOIO
      wa_properties-name = if_srm_document=>prop_document_id.
      wa_properties-value = i_document_id.
      APPEND wa_properties TO lt_properties.

      wa_properties-name = if_srm_document=>prop_description.
      wa_properties-value = 'DESCRIPTION'.
      APPEND wa_properties TO lt_properties.

** Generate document
      CALL METHOD me->if_srm_generic_sp~create_document
        EXPORTING
          properties     = lt_properties
          do_commit      = srmgs_false
          do_update_task = srmgs_false.


** Get reference to IF_SRM_DOCUMENT
      CALL METHOD me->if_srm_generic_sp~get_document
        IMPORTING
          document = lif_document.

** Apply lock
      lif_document->apply_lock( ).

** Set document ID and description for Phio
      REFRESH lt_properties.

      wa_properties-name = if_srm_document=>prop_document_id.
      wa_properties-value = i_document_id.
      APPEND wa_properties TO lt_properties.

      wa_properties-name = if_srm_document=>prop_description.
      wa_properties-value = 'DESCRIPTION'.
      APPEND wa_properties TO lt_properties.

** Generate variant
      CALL METHOD lif_document->new_variant
        EXPORTING
          properties            = lt_properties
          create_new_log_version = srmgs_false
        RECEIVING
          new_variant           = lif_variant.
```

```
** Set content
      lif_tab_cont ?= lif_variant.
      CALL METHOD lif_tab_cont->set_content
        EXPORTING
          components    = component_tab
          ascii_content = ascii_content.

** Remove lock
      lif_document->remove_lock( ).

** Exception handling
    CATCH cx_srm_gsp INTO ex_gsp.

      CASE ex_gsp->error_type.
        WHEN cx_srm_gsp=>duplicate_objid.
              … .
        WHEN cx_srm_gsp=>internal_error.
             … .
        WHEN cx_srm_gsp=>not_authorized.
             … .
        WHEN cx_srm_gsp=>parameter_error.
             … .
      ENDCASE.

  ENDTRY.
```

2.3.2 Opening a Document and Reading its Content

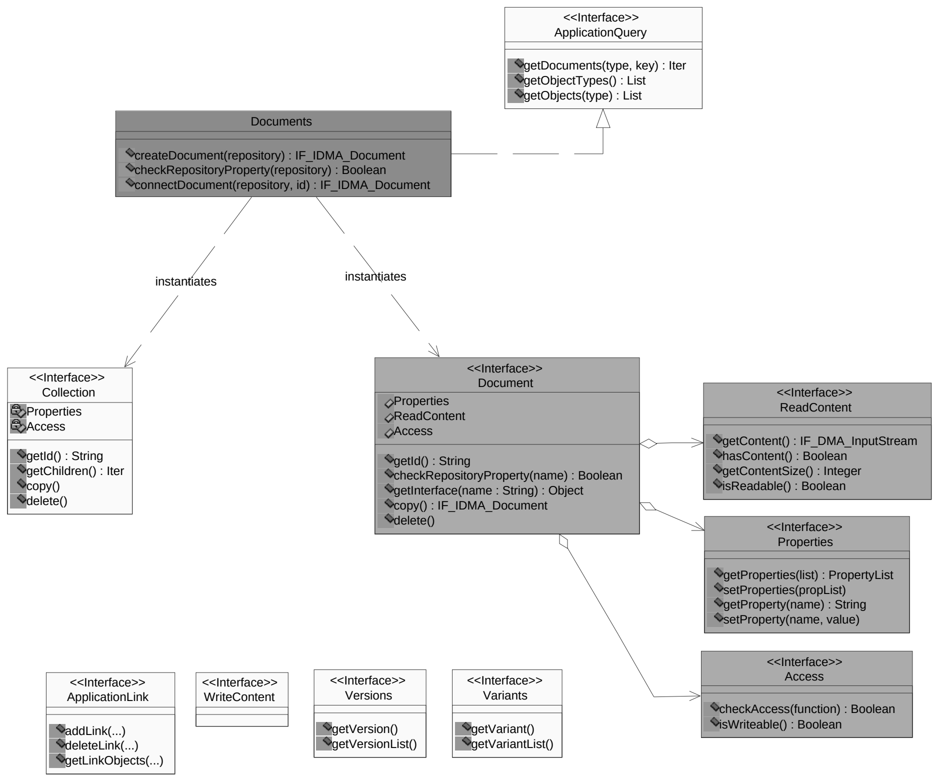
Coding Example

```
  DATA:   lif_document           TYPE REF TO if_srm_document,
          lif_variant            TYPE REF TO if_srm_variant,
          lif_phio_properties    TYPE REF TO if_srm_gsp_properties,
          lt_properties          TYPE srmgs_property_tab,
          wa_properties          TYPE srmgs_property,
          l_boolean              TYPE srmgs_boolean,
          lif_tab_cont           TYPE REF TO if_srm_gsp_tab_transfer,
          lt_component           TYPE srmgs_components,
          ascii_content          TYPE srmgs_ascii_content,
          ex_gsp                 TYPE REF TO cx_srm_gsp.

  TRY.

** Get reference to IF_SRM_DOCUMENT
      CALL METHOD me->if_srm_generic_sp~get_document
        IMPORTING
          document = lif_document.

** Get required variant of the document
      CALL METHOD lif_document->get_variant
        EXPORTING
          version_id = '0'
          variant_id = '0'
        RECEIVING
          my_variant = lif_variant.

** Get attribute interface of the variant
      lif_phio_properties = lif_variant->get_property_interface(  ).

** Get certain attribute values of the variant
      wa_properties-name = if_srm_document=>prop_document_id.
      APPEND wa_properties TO lt_properties.

      wa_properties-name = if_srm_document=>prop_description.
      APPEND wa_properties TO lt_properties.

      wa_properties-name = if_srm_document=>prop_props_changed_by.
      APPEND wa_properties TO lt_properties.

      CALL METHOD lif_phio_properties->get_properties
       IMPORTING
           props_deleted = l_boolean
      CHANGING
           properties = lt_properties.


** Get content (by internal table)
      lif_tab_cont ?= lif_variant.
      CALL METHOD lif_tab_cont->get_content
        IMPORTING
          components    = lt_component
          ascii_content = ascii_content.

** Exception handling
    CATCH cx_srm_gsp INTO ex_gsp.

      CASE ex_gsp->error_type.
        WHEN cx_srm_gsp=>internal_error.
             … .
        WHEN cx_srm_gsp=>not_authorized.
             … .
        WHEN cx_srm_gsp=>parameter_error.
             … .
      ENDCASE.

  ENDTRY.
```

3 Using the Generic Document Management API (GDMA)

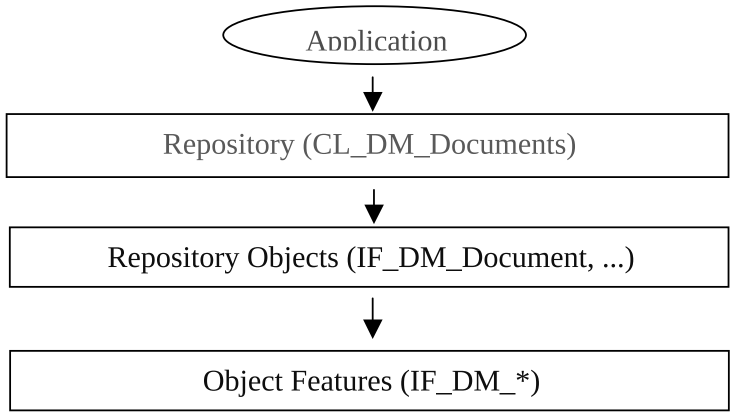
3.1 Overview

Documents are implemented in this system as objects that have different services to offer. One document is represented by the interface “IF_DM_Document“. This interface can be used to access all other features (interfaces) of this document.
The following UML diagram shows how some of the most important interfaces and objects work together.



The main interface also brings with it three further interfaces which are important for the usage of a document. These interfaces are avail able as attributes on the main interface. Further interfaces are available by calling "getInterface". Before you have a document, it must be instantiated from a repository. A repository is represented by an object of the class "CL_DM_Documents". These obje cts are generic, which means that two different objects of this class can represent totally different repositories with different properties. This class is the entry point for the application developer.

The application developer has the following view of the whole document infrastructure:



Application
Repository (CL_DM_Documents)
Repository Objects (IF_DM_Document, ...)
Object Features (IF_DM_*)

3.2 Exceptions

General information about exceptions:
The methods in the GDMA can trigger different exceptions. There are two main exceptions classes from which all other exceptions that are triggered should be inherited. The first one is CX_DM. This class inherits from CX_STATIC_CHECK. It is used for all exceptions that must be checked at the first point that they occur. The other class is CX_DM_NO_CHECK, which inherits from CX_NO_CHECK. This class is to be used for all exceptions where an explicit check would be overkill. For example, in the case of CX_DM_PARAMETER_ERROR, if you were to check this exception at all possible occurrences, the applications would become totally cluttered. Furthermore, this exception can only occur in faulty programs – and that should not happen in the finished program. It is also impossible to fix the results of a programming error in the program (if it was possible, why not fix the error in the first place?).
The GDMA exceptions have some special properties which must be provided by the method that triggers the exception. These can also be used in the error description text:

Exception Properties		
Name	Data Type	Semantics
Repository	String	ID of the repository in which the problem occurred.
SRC_CLASS	String	Implementing class (of SRC_INTERFACE) in which the problem was encountered.
SRC_INTERFACE	String	GDMA interface that was called.
SRC_METHOD	String	Interface method that was called.
SRC_DOCID	String	ID of the document where the problem occurred (if any – else empty).
REASON	String	**This property is optional.** Here you can give a reason for the problem. This may be a HTTP error code/message, an (English) description, or anything else that helps to pinpoint the problem.
ADD_INFO	String	**This property is optional.** Additional info. Here you can state the ID of a troublesome object, for example.
REPOSITORY	String	ID of the repository (DPS-ID) where the problem occurred (if any – else empty).

This framework provides a set of exceptions that can be used for error conditions. For the person implementing the interfaces, it is also possible to define more detailed exception classes. These must inherit from the standard exception classes. It is also discouraged to add new direct subclasses to one of the two base classes CX_DM and CX_DM_NO_CHECK. Instead, you should subclass one of the classes that are yet to be used in the interface you are implementing. In this way, you are not adding descriptions, rather are only describing the error condition more precisely.

Here is the list of the standard exceptions:

Standard Exceptions			
	Name	Semantics	Check
	CX_DM	Base class for checked exceptions (inherits from CX_STATIC_CHECK). Please don't subclass directly!	X
	CX_DM_NO_CHECK	Base class for unchecked exceptions (inherits from CX_NO_CHECK). Direct subclassing of this class is discouraged.	
1	CX_DM_ACCESS_DENIED	Access to a resource is denied. A more precise description must be given, at least in the text.	X
2	CX_DM_INTERFACE_NOT_AVAILABLE	An interface that was requested is not available.	X
3	CX_DM_NOT_FOUND	A requested resource was not found.	X
	CX_DM_PARAMETER_ERROR	A method was called with an invalid parameter (parameter combination).	
4	CX_DM_READ	An error occurred while reading information from the resource store.	X
5	CX_DM_WRITE	An error occurred while writing information to the resource store.	X
6	CX_DM_READ_WRITE	An error occurred while reading or writing information from or to the resource store. **ATTN**: This class is a super-class for CX_DM_READ and CX_DM_WRITE. If it is possible to decide which error occurred, the specialized exception should be raised instead of this one.	X
7	CX_DM_NOT_AVAILABLE	Service is not available	X
	CX_DM_UNEXPECTED	An unexpected error occurred.	

The numbers for the exceptions that are triggered by the methods are added after the signature in curly parentheses and this color.

3.3 Access to Documents

Every document is represented by a unique string – its (technical) document ID. The format of this ID may va ry from content model to content model. For now, there is only one restriction: the document ID should not be longer than 255 characters.
A repository is represented by an object of the class CL_DM_Documents.

CL_DM_Documents	
Attributes	
Access : IF_DM_Access	Interface for access rights. With this interface it is possible to check access rights that are not dependable of a specific document. E.g. search rights.
Methods	
CONSTRUCTOR(docProviderSpace: CSequence) {3}	Constructor. The document provider space defines the repository for the documents.
connectDocument(ID: CSequence) {1, 3, 4}	Connect a document using its ID.
getNewDocId() : String {1, 5, 7}	Get a new document ID that can be used to provide for createDocument. This method does not create a new document but only reserves a valid ID that can be used later for a new document.
createDocument(*ID : String*) : IF_DM_Document {1, 6, 7}	Create a new document The ID is optional and must be an ID obtained from getNewDocId.
getInterface(intf: ref to object) {2}	Get the implementation of one special interface.
getPropertyDefinitions() : PropertyDefTab	Get definitions of document properties. This method allows access to the definitions of possible document properties without having a document.
getRepositoryProperty(name: CSequence) : String	Get a property of the repository (see 3.10)
setRepositoryParameter(name: CSequence, value: CSequence) {7}	Set a parameter of the repository (see 3.11).
Convenience Methods	
Note: *Italic* parameters are optional.	

When you create a new object of this class, you supply a parameter docProviderSpace. This parameter is somewhat comparable to the element type of the records managem ent framework. It defines the repository where the documents are to reside. All information that is further needed for the repository, is stored in the framework registry.

3.4 Basic Functions

There is one main interface that represents a document resource: IF_ DM_Document. According to the first diagram, three other interfaces are easily reached from this interface:

• IF_DM_Properties	- access to the properties of a document
• IF_ReadContent	- reading of the document content
• IF_DM_Access	- checking the access rights to the document

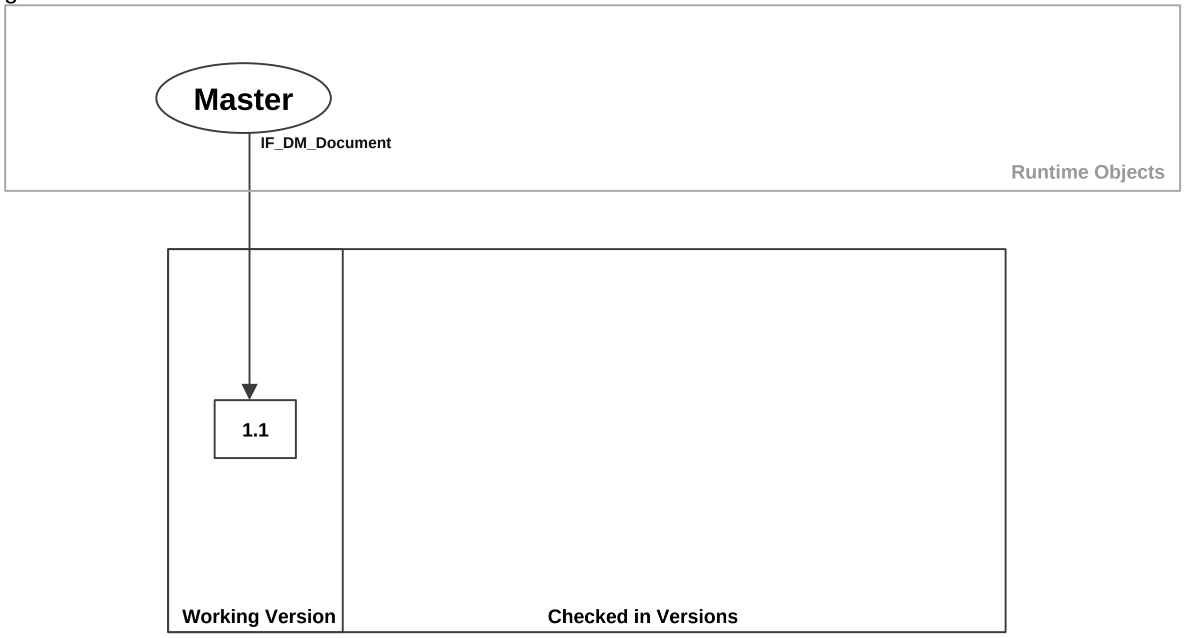
IF_DM_Document	
Attributes	
Properties : IF_DM_Properties	Properties Interface
ReadContent : IF_DM_ReadContent	ReadContent Interface
Access : IF_DM_Access	Access Interface
Methods	
getId() : String	Get document ID
getInterface(intf: ref to object) {2}	Get the implementation of one special interface
getRepositoryProperty(name: CSequence) : String	Get a property of the repository (see 3.10)
getRepository() : CL_DM_Documents	Get object that represents the repository
copy(newId: String) : IF_DM_Document {1, 6, 7}	Create a copy of the document. The newId is optional and must be an ID obtained from getNewDocId.
delete() {1, 6, 7}	Delete document
Convenience Methods	
isWritable() : Boolean	Check whether document is writable

All other interfaces that may be supported can be accessed by calling the method getInterface. It is also possible to get the repository object by calling getRepository. In this way, it is possible to transfer a document to a service and for this service to also have access to the repository. In this way, document handling is made a lot easier.

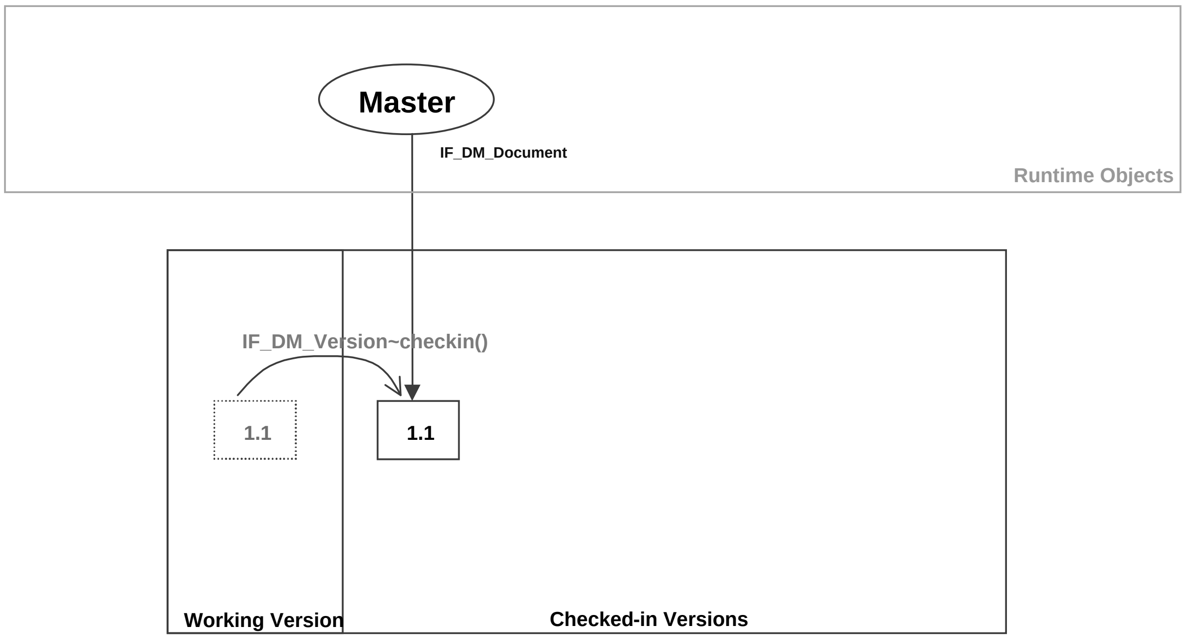
3.5 Versioning

This API uses a specific model for versioning. One problem is that both non -versioning repositories and repositories th at support versioning must be supported. I decided to use a model similar to that of WebDAV, because this model also supports both types of repositories.

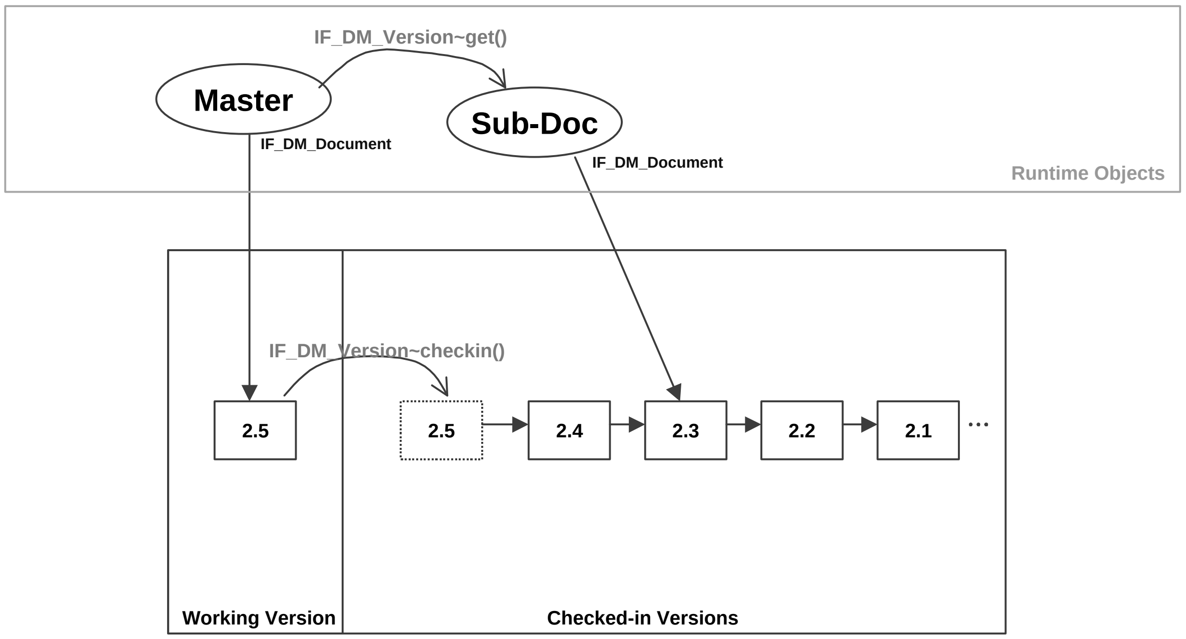
In this model, one document can be divided into several parts. The first part is the working version – it is created after a document is created and the content is written the first time. The interface IF_DM_Document gives you direct access to this version. The interface object is called the master document. When you access a document you always get the ma   ster document first. You can then use that to access all other parts of the document as they become available. If you have no versions and no variants, the master document is all you get.



If a repository supports versioning, then a version history can b e created by calling the check-in method.



The checked-in versions are a linear list of versions that cannot be modified. It is possible to access such versions with the get method.



The standard way to modify versions after the first check-in is to check out the latest checked-in version and then modify this working version. Some repositories may support auto-checkout, but it is better to not rely on this.
The master document represents either the working version (if there is one and you are the owner) or the most recently checked -in version of the document. A repository may decide if a non-owner of the working version sees the working version or only the most recently checked-in version. This may prevent others from viewing unreleased versions.

This interface is provided for versioning:

IF_DM_Version	
Methods	
get(ID: CSequence) : IF_DM_Document {1, 3, 4}	Get a specific version
getList() : ObjectTab {1, 4}	Get list of existing versions
checkout() {1, 6, 7}	Check out the current version and set it as the working version. This can only be done if there is no working version yet. This method must be called if the repository property AutoCheckout is false. If the property is true, you can still “manually” check out by calling this method.
checkin() {1, 6, 7}	Check in the current working version of the document. This is then the new “current” version.
Convenience Methods	
getActualVersion() : IF_DM_Document {1, 3, 4}	Get the most recent checked-in version. If only the working version exists, this method fails. The resulting document is read-only.
Note: The (master) document itself represents either the current working version or the most recent checked-in version.	

3.6 Variants

Variants are a special type of versions. Variants of a document can be a translation (language variant), or another format version of the same content. The essential difference is that variants represent the same development state of the document and in essence the same semantic.
The variant that you acce ss directly in the master document or when you access a specific version, may depend on your language and further settings. When you create a new version of a document (or a new document), it is automatically created as an OR (original) variant, meaning yo u can have variant support without using this interface. Later you might want to add other variants using this interface.
The variant interface should be accessed after the resolution of the correct version of a document.

IF_DM_Variant	
Methods	
get(ID: CSequence) : IF_IDMA_Document {1, 3, 4}	Get a specific variant
getList() : ObjectTab {1, 4}	Get list of existing variants
create(tag: CSequence, id : CSequence) : IF_IDMA_Document {1, 6}	Create a new variant. The tag specifies the type of variant. The parameter ID does not need to be evaluated if the repository supports no IDs (see Property MaxVariantId). Variant Tags OR Original variant LA Language variant FO Format variant The first (original) variant must not be created because it is implicitly created when a new version (or new document) is created. A new variant that is created by this method starts without any content (no components). Some properties may have been copied or implicitly created.

3.7 Access Control

3.7.1 Locking

The locking interface mus t be supported by all repositories that support the changing of documents.

IF_DM_Lock	
Methods	
lock(lockUser: String) : Boolean {1, 6}	Lock the resource. If the resource is still locked, False is returned and lockUser is filled with the ID of the locking principal. If the lockUser is not known, the method returns the value "<<unknown>>".
unlock() {6}	Release the lock for the resource

3.7.2 Access Rights

The access rights for one operation must be checked by the repository when the operation is requested. However, it is also possible for an application to check in advance if a privilege is available to the current user.

IF_DM_Access	
Methods	
checkAccess(privilege: CSequence) : Boolean	Check if current user has a specific privilege

The access interface is a vailable on CL_DM_Documents and on every document itself. On CL_DM_Documents, privileges that are not bound to one specific document can be checked.

3.7.3 Resource-Dependent Privileges

Privileges			
Identifier	Contains	M	Semantics
all	read, write, unlock-foreign		
read	read-properties, read-content, read-acl	X	Read privilege for resource
write	write-properties, write-content, write-acl	X	Write privilege for resource
read-data	read-properties, read-content		Read privilege only for properties and content
write-data	write-properties, write-content		Write privilege only for properties and content
create	create-content, create-properties		Create elements of a resource
delete	delete-content, delete-properties		Delete elements of a resource
unlock			Unlock a resource that was locked by **another** principal
read-properties			read properties of resource
write-properties	delete-properties, create-properties, change-properties		(over)write properties of resource
create-properties			The privilege to create (formerly) non-existing properties. *(Non-existing means that the property had no value on this resource)*
delete-properties			Delete existing properties (meaning, delete the value of a property)
change-properties			Change the value of an existing property, without the need of creating a new version.
read-content			read content of a document
write-content	delete-content, create-content, change-content		(over)write content of a document
create-content			Create new content
delete-content			Delete existing content
change-content			Change existing content, without the need of creating a new version.
read-acl			Read access control list of a resource
write-acl			Write access control list of a resource
All privileges above must be understood by a valid implementation. Not all implementations have to support these privileges, but they have to at least react in a defined and useful way to calls of **IF_DM_ACCESS~ checkAccess(***privilege***)**, where *privilege* is one of the above. At least the mandatory (column "M") privileges must be supported. If one privilege is not supported, then the privileges must be handled as always true or always false. For example, if there is no special privilege for properties, then a check for read-property will be true if the privilege read or read-content is set. If there is no support for unlocking of somebody else, unlock will always return false.			

3.7.4 Resource-Independent Privileges:

Resource-Independent Privileges are privileges which are not set for one specific resource but for all resources in one c ontent space. They are set using the same interface IF_DM_Access, but the interface must be located at the object representation for the content space (CL_DM_Documents).
Property Groups are collections of properties which are assigned to the group. The gro ups are a helpful resource to simplify setting the privileges. There is only one predefined group – the group "all" which includes all properties. Property groups are an optional feature that can be – but does not have to be – supported by an implementatio n of a repository. The support for property groups is independent of the support for ACLs. The old KPro -SP implementations (Generic SP 1.0 and Warp 3) were also capable of property groups.

Independent Privileges			
Identifier	Contains	M	Semantics
create-document			create a new document
create-collection			create a new collection
search			search documents
read-propgroup-<propgroup>			read accessibility for all properties of one specific property group (all documents).
write-propgroup-<propgroup>			write accessibility for all properties of one specific property group (all documents).

3.7.5 Access Control Lists (ACL)

The access interface is rather independent of the implementation of access control. One example of this is access control lists (ACLs). The privile ges used here are exactly the same as described above.
Below, a principal can be a user or a group of users that can have privileges.

IF_DM_AccessControl	
Methods	
getACL() : IF_DM_ACL {1}	Get the ACL of a resource
setACL(acl: IF_DM_ACL) {1}	Set an ACL for the resource
getSupportedPrivileges() : PrivilegeTab	Get all the privileges that are supported by this resource / repository.

In the PrivilegeTab there are pointers to the interface IF_DM_Privilege:

IF_DM_Privilege	
Methods	
getIdentifier() : String	Get the identifier of the privilege
getDescription(): String	Get the description for the privilege. The description given should be in the login language of the current user.
getSubPrivileges() : PrivilegeTab	Get a table of aggregated privileges which are contained in this.

IF_DM_ACL	
Methods	
getPrincipals() : ObjectTab	Principals which are known in the ACL.
getPrivilegeSet(principal: CSequence, granted: ObjectTab, denied: ObjectTab)	Get granted and denied privileges for one principal. The set contains only the direct privileges of the principal – privileges that are granted using a group membership are not given.
setPrivilegeSet(principal: CSequence, granted: ObjectTab, denied: ObjectTab)	Set granted and denied privileges for one principal. Inherited privileges or privileges by membership are not affected (but may be overshadowed).
Note: Underlined parameters are output parameters.	

3.8 Properties

3.8.1 Property Access

One of the main interfaces for a document is the one for property access. It is possible t o set property values and to get property values.

IF_DM_Properties	
Methods	
getProperties(list: ObjectTab) : PropertyTab {1, 4, 7}	Get document properties. If there is no access to one or more properties, they are not delivered.
setProperties(properties: PropertyTab) {1, 5, 7}	Set document properties
getPropertyDefinitions() : PropertyDefTab {1, 4, 7}	Get definitions of document properties. The resulting table contains pointers to the interface IF_DM_PropertyDef.
deleteProperties(list: PropertyTab) {1, 5}	Delete document properties. If properties have multiple values, then only the given values are deleted. If the property can only have one value, that value will be deleted even if it does not correspond to the given value (which is ignored).
clearCache()	Clear a property cache, if it exists. A user can call this method if he knows that data has been changed by another task **or** if he wants to make absolutely sure of getting the current data. If no property cache exists, the implementation of this method may be empty.
Convenience Methods	
getProperty(name: CSequence) : String	Get one property
setProperty(name: CSequence, value: CSequence) {1, 5, 7}	Set one property

This Interface also allows you to access the definition of the properties for documents of this type.

3.8.2 Property Definitions

The property definitions interface rests on the model that every property definition supplies meta properties, which describe the specific property.

IF_DM_PropertyDef	
Methods	
getDefProperty(name: CSequence) : String	Get a property value of a property
Convenience Methods	
getIdentifier() : String	Get the identifier of a property
getDescription() : String	Get description of a property
getDefBoolProperty(name: CSequence) : Boolean	Get a Boolean property of a property
getDefIntProperty(name: CSequence) : Integer	Get an integer property of a property

How the property definitions are customized is not subject of this specification.

Property Definitions		
Name	Data Type	Semantics
Name	String	Name of the attribute
Description	String	Description text (2)
IsModifiable	Boolean	Property may be changed (1)
IsModifiableOnce	Boolean	May be changed only once
IsModifiableViaSelect	Boolean	Changed only by a selection dialog
IsDeleteable	Boolean	Property value may be deleted (1)
IsMandatory	Boolean	Mandatory property
IsUnique	Boolean	Property can only have one value
IsHidden	Boolean	Will not be shown (also no columns in tabs)
IsInheritFromPreVersion	Boolean	Will be inherited from version to version
IsLanguageSensitive	Boolean	Property value is language-dependent
IsQueryable	Boolean	Property query is possible (1)
ListPosition	Integer	Position in a list view
ValueCheck-Table	String	DDIC table for value check
ValueCheck-Field	String	DDIC field for value check
Extensibility: Everybody may define additional attributes. However, the names must then start with “<module>_“ as a prefix. The prefix must have at least 3 uppercase letters. **Remarks:** (1): May be modified by access rights. Additionally, it may be possible that the values of properties are not given by getProperties, if no read access exists for them (this is not related to IsHidden). (2): May be language-dependent.		

3.9 Content Access

3.9.1 Standard Content Access Methods

The standard method for content access is access through r eading/writing the content as byte-stream through the memory of the application server. This method is more convenient. Every repository must support this type of content access. If the content is only readable, then only IF_DM_ReadContent must be supporte d. If the content is also writable, then IF_DM_WriteContent must be supported as well.

IF_DM_ReadContent	
Methods	
read(offset: Numeric, length: Numeric) : XString {1, 4}	Get document content
GetSize() : LongInt (N16) {1, 4}	Get size of content
getMimeType() : String {1, 4}	Get MIME type content
getEncoding() : ABAP_ENCODING {1, 4}	Get encoding of content
Convenience Methods	
hasContent() : Boolean	Is there content?
isReadable() : Boolean	Is the content (for current user) readable?
Note: This interface must be supported by any content repository.	

IF_DM_WriteContent	
Methods	
setMimeType(mimeType: CSequence)	Set the MIME type of the document. If the MIME type could not be set, the error is ignored – the error must be reported during calls of write or finishWrite.
setEncoding(encoding: ABAP_ENCODING)	Set the encoding of the document. If the encoding could not be set, the error is ignored – the error must be reported during calls of write or finishWrite.
write(content: XString, position: LongInt, size: integer) {1, 5}	Set document content. This method can be called as often as needed, but should be called at least once even for empty content.
finishWrite() {5}	Finish the write. *setMimeType*, *setEncoding* and *write* must be called before calling this.
Convenience Methods	
IsWritable() : Boolean	Is the content (for current user) writable?
Note: This interface must be supported by any *writable* content repository.	

3.9.2 Content access via URL

Sometimes the access by URL is more convenient. Particularly when visualization components are available which can handle the URLs. However, this type of interface includes a number of insecurities, because the URLs can be generated by different sources and not every consumer may be able to handle all URLs. Neverth eless, it is also one way to avoid streaming the whole content through the application server.

IF_DM_ReadURL	
Methods	
getReadURL(type: UrlKind, location: LocationInfo) : String {1, 4}	Get URL for content. The location (of the destination for the content) can be used by the implementation to optimize the routing of the data, because front-end and back-end of the system may be in remote places.
getSize() : Integer {1, 4}	Get size of content
getMimeType() : String {1, 4}	Get MIME type content
getEncoding() : ABAP_ENCODING {1, 4}	Get encoding of content
Convenience Methods	
hasContent() : Boolean	Is there content?
isReadable() : Boolean	Is the content (for current user) readable?
Note: This interface does not necessarily need to be supported. There may also be a standard implementation that maps IF_DM_ReadContent to this one. This may result in poor performance, because the data must be routed through the application server (location will be ignored).	

IF_DM_WriteURL	
Methods	
setMimeType(mimeType: CSequence)	Set the MIME type of the document
setEncoding(encoding: ABAP_ENCODING)	Set the encoding of the document
getWriteURL(type: UrlKind, location: LocationInfo) : String {1}	Get URL for writing operation. Here we are also able to give a location (in this case, location of the content source) that may be helpful for the routing of the data (see IF_DM_ReadURL->getReadURL).
finishWrite(){5}	Finish the write. *setMimeType*, *setEncoding* and *getWriteURL* (+ writing to it) must be called before calling this.
Convenience Methods	
isWritable() : Boolean	Is the content (for current user) writable?

3.9.3 Content Access by File

Like URL access, the access by file may have advantages. Moreover, it could enable you to minimize the data transfer.

IF_DM_ReadFile	
Methods	
download(file: String, location: LocationInfo) {1,6}	Create file with the content of the document. The location can be used by the implementation to optimize the routing of the data.
getSize() : Integer {1, 4}	Get size of content
getMimeType() : String {1, 4}	Get MIME type content
getEncoding() : ABAP_ENCODING {1, 4}	Get encoding of content
Convenience Methods	
hasContent() : Boolean	Is there content?
isReadable() : Boolean	Is the content (for current user) readable?
Note: This interface does not necessarily need to be supported. There may also be a standard implementation that maps IF_DM_ReadContent to this one. This may result in poor performance, because the data must be routed through the application server (location will be ignored).	

IF_DM_WriteFile	
Methods	
setMimeType(mimeType: CSequence)	Set the MIME type of the document
setEncoding(encoding: ABAP_ENCODING)	Set the encoding of the document
upload(file: String, location: LocationInfo) {1, 3, 6}	Set document content through a file content
finishWrite() {5}	Finish the write. *setMimeType*, *setEncoding* and *upload* must be called before calling this.
Convenience Methods	
isWritable() : Boolean	Is the content (for current user) writable?

3.9.4 Components

Some repositories may also support compo nents. Components are also treated as versions. You must only access the component interface after the resolution of version and variant.

IF_DM_Component	
Methods	
get(ID: CSequence) : IF_DM_Document {1, 3, 4}	Get a specific component
getList() : ObjectTab {1, 4}	Get list of existing components
create(id: CSequence, *compNum: Integer*) : IF_DM_Document {1, 6}	Create new component. The parameter ID does not need to be evaluated if the repository does not support IDs (see Property MaxComponentId). The first (main) component is created when you first save content on a document(/version). You only need to create additional components with this method. The parameter compNum is optional and does not need to be evaluated if the repository does not support it (see Property CompNumSupported).
getCompNum(ID: CSequence) : Integer	Get the component number of a component if the repository supports this feature. If not, then return -1.
Note: *Italic* parameters are optional.	

3.10 Repository Properties

Repository properties are read-only properties of a repository which can be accessed through getRepositoryProperty on the interface IF_DM_Document, or on an object of the class CL_DM_Documents.
With these properties you are able to check special features of the repository.

Repository Properties		
Name	Data Type	Semantics
ClientDependant	Boolean	Repository is client-dependent
DoesCommit	Boolean	Changes are committed immediately – no rollback possible. The value of this property may change if the repository supports the repository parameter DoCommit (see 3.11).
ACL	Boolean	Supports ACLs (and also the ACL interface)
Create	Boolean	Creation of documents
PropGroups	Boolean	Supports property groups
LocalCache	Boolean	Supports local (optimized) caching of documents.
AutoCheckout	Boolean	When a document is written to, then an automatic checkout is performed.
ShowWorkingVersion	Boolean	When this flag is set, a read on the master document always retrieves the current working version of the document when there is one available. If no working version is available, the current version is always taken. If it is not set, a read retrieves the current version of the document unless the current user is the user who performed the checkout or created the document.
MaxVariantId	Integer	Maximum size of (user-defined) variant IDs. If no user-defined variant ID is possible, the value is 0.
MaxComponentId	Integer	Maximum size of (user-defined) component IDs. If no user-defined component ID is possible, the value is 0.
InUpdateTask	Boolean	Writes data in update task to the repository.
IsTransportable	Boolean	Documents in this repository are transportable.
SubSpaceId	String	The ID of the document subspace. See repository parameter SubSpaceId. Not all repositories support subspaces. If not, they always return an empty string here.
CompNumSupported	Boolean	The repository supports component numbers, if this property is True.

3.11 Repository Parameters

Repository parameters are values that can be set, which influence the operation of the repository. Repository parameters can be changed by calling setRepositoryParameter on an object of the class CL_DM_Documents.

Below is a list of standard repository parameters. A repository does not need to support any parameter. There may also be repository-specific parameters. For such special parameters, the naming convention of property definitions also applies (<module>_...).

If a parameter is not supported, the exception CX_DM_NOT_AVAILABLE must be triggered.

Repository Parameters		
Name	Data Type	Semantics
Login	String	A login name/ID that may be required for logins on the repository
Password	String	A password that may be required in addition to Login.
DoCommit	Boolean	Change the commit behavior. If this parameter is changed, the value of the repository property DoesCommit changes accordingly.
InUpdateTask	Boolean	Change repository property “InUpdateTask” – if possible.
SubSpaceId	String	It is possible to divide one document provider space into subspaces, and to provide along with this parameter the ID of a subspace that is to be used.

3.12 Query

For the execution of queries, two different interfaces are possible, because different repositories may have different capabilities. In the R/3 area, the selection options are very popular. With these it is possible to handle many types of queries, however there are some queries that cannot be modeled with this technique. This is the reason for the second interface – the operation query. This allows you to use a Boolean algebra type of query description that is much more general than selection options. A repository may choose to implement one of these interfaces or both. It is also possible to generically implement the selection options interface based on the operation query interface. This implementation can be added to any service provider that implements the more general interface on its own.

3.12.1 Query Interface

IF_DM_Query	
Methods	
execute(desc : QueryDescTab, maxHits: Integer, searchProps: PropertyTab, resultProps: ObjectTab, <u>result: ResultList, props: ObjectPropTab</u>)	Execute a query that is defined by the given query description table. If the query description is invalid, CX_DM_PARAMETER_ERROR must be raised. searchProps can consist of the following (depending on which properties are supported by the repository):
Convenience Methods	
Note: <u>Underlined</u> parameters are output parameters.	

search Properties	
LANGUAGE	Language in which to search
ONLY_ACTUAL	Search only in actual versions
NEAR_BY	Maximum word distance of multiple phrases
RMS_ID	RMS-ID (Records Management)
SPS_ID	SPS-ID (Records Management)

Structure QueryDescTab

Name	Data Type	Semantics
GroupNo	Integer	The number of a group of atomic operations (one line in the table is one atomic operation). The entries in this table must be sorted by this group number. The union of all results of any atomic operation in one group will be created. The results of different groups will be intersected.
PropertyId	STRING	Name of the property or left empty for full-text search.
Operator	STRING	Possible values:

Option Values:	
EQ	Property equal to Value
NE	Property not equal to Value
LT	Property less than Value
GT	Property greater than Value
LE	Property less than or equal to Value
GE	Property greater than or equal to Value
BT	Property between Value and HighValue
NB	Property not between Value and HighValue
CP	Contains Pattern
NP	No Pattern

Name	Data Type	Semantics
Value	STRING	Value for comparisons. It serves as low value when option is BT
HighValue	STRING	High Value when option is BT, otherwise ignored.

Structure QueryResultList

Name	Data Type	Semantics
ResultNum	Integer	Unique result number
DocId	String	The ID of the document found.
Version	String	The version ID of the document (optional)
Description	String	Descriptive Text for the document, which can be displayed in a results list (optional). The repository can decide which property may be used for this information.

Structure ObjectPropTab		
Name	Data Type	Semantics
ResultNum	Integer	The ID of the result in the QueryResultList.
PropName	String	Name of the property
PropValue	String	Value of the property

Examples:

```
1|PROP1|EQ|C*      selects all documents with PROP1 = "C*"

1|PROP1|CP|C*      selects all documents with PROP1 matching C*

1|PROP1|EQ|JOHN    selects all documents with PROP1 = "JOHN"
1|PROP1|EQ|ANNA      OR PROP1 = "ANNA"

1|PROP1|EQ|JOHN    selects all documents with PROP1 = "JOHN"
2|PROP1|EQ|ANNA      AND PROP1 = "ANNA" (if PROP1 has multiple values)

1|PROP1|EQ|JOHN    selects all documents with PROP1 = "JOHN"
1|PROP2|EQ|ANNA      OR PROP2 = "ANNA"

1|PROP1|EQ|JOHN    selects all documents with PROP1 = "JOHN"
2|PROP2|EQ|ANNA      AND PROP2 = "ANNA"

1|PROP1|EQ|JOHN    selects all documents with PROP1 has one value "BERT"
1|PROP2|EQ|ANNA      AND    either has PROP1 another value "JOHN"
2|PROP1|EQ|BERT         OR PROP2 = "ANNA"
```

4 Implementing a GDMA Service Provider

To implement a GDMA Service Provider, proceed as described below.

4.1 Repository Connection

Declare a class that inherits from the basis cla   ss CL_DM_REP_IMPL. Implement the repository connection in this class.

The class CL_DM_REP_IMPL has the interface IF_DM_REP_IMPL.

4.1.1 IF_DM_REP_IMPL

Some of the interface methods have been implemented already, and some you still have to implement. The following list gives you an overview:

Method Name	Explanation	Implementation
GET_NEW_DOC_ID	Returning DOC_ID You generate a new document ID for a new document.	To be implemented
CREATE_DOCUMENT	Importing IMP_DOC_ID (optional), Returning DOC_ID. You generate a new document.	To be implemented

	The caller of the GDMA has the option of handing over a DOC_ID. If the caller does not hand over a DOC_ID, you have to generate one. You do this by calling IF_DM_REP_IMPL~GET_NEW_DOC_ID( ), see above. The method is called within the method CREATE_DOCUMENT( ) of class CL_DM_DOCUMENTS.	
GET_PROPERTY_DEFINITIONS	Returning PROPERTY_DEFS. You return the properties of the document attributes. The method is called within the method GET_PROPERTY_DEFINITIONS( ) of class CL_DM_DOCUMENTS.	To be implemented
GET_REPOSITORY_PROPERTY	Importing NAME, Returning VALUE. You receive the name of a repository attribute and return the value of the required repository attribute. Repository attributes are attributes that return information about the repository. You can find a standard for repository attributes in the list on page 32. We recommend that you use these attributes. The method is called within the method GET_REPOSITORY_PROPERTY( ) of class CL_DM_DOCUMENTS.	To be implemented
SET_REPOSITORY_PARAMETER	Importing NAME, VALUE. You receive a name-value pair of a repository parameter and usually set this as class attribute for your class. Repository parameters are attributes that can change the behavior of the repository. You can find a standard for repository attributes in the list on page 33. The method is called within the method SET_REPOSITORY_PARAMETER( ) of class CL_DM_DOCUMENTS.	To be implemented
CONNECT	Among other tasks, sets class attribute MY_REPOSITORY. This class attribute includes an object of the class CL_DM_DOCUMENTS that represents the repository.	Implemented already. Cannot be redefined.
GET_REPOSITORY	The repository is read from the class attribute MY_REPOSITORY.	Implemented already. Cannot be redefined.

In addition, the class CL_DM_REP_IMPL has the following methods:

Method Name	**Explanation**	**Implementation**
CHECK_CONNECT	You check whether the specified repository exists. For more details, read the comment in the method.	Must be redefined.
GET_CONNECTION_PARAMETER	Returns connection parameter.	Implemented

		already. Cannot be redefined.

4.2 Document Services

Declare a class that inherits from CL_DM_DOCUMENT. In this newly declared class, you implement the services which are to be made available for document processing.

The class CL_DM_DOCUMENT has the interface IF_DM_DOCUMENT.

4.2.1 IF_DM_DOCUMENT

This interface represents a document. Some of the interface methods have already been implemented, some of them still have to be implemented. The following list gives you an overview:

Method Name	**Explanation**	**Implementation**
COPY	Importing: NEW_ID (optional), Returning NEW_DOC You receive the new ID, which was generated using IF_DM_REP_IMPL~GET_NEW_DOC_ID. You copy the current document and return this as an interface reference to IF_DM_DOCUMENT.	To be implemented
DELETE	You delete the current document.	To be implemented
IS_WRITABLE	Returns an indicator for whether the current document is allowed to be changed. Internally, the method CHECK_ACCESS is called on the interface IF_DM_ACCESS. That is where you perform the actual implementation.	Implemented already. Cannot be redefined.
GET_INTERFACE	Returns a reference to the interface which the caller used to type the parameter.	Implemented already. Cannot be redefined.
GET_REPOSITORY_PROPERTY	Returns the value for a repository attribute. Internally, the method GET_REPOSITORY_PROPERTY is called on the interface IF_DM_REP_IMPL. That is where you perform the actual implementation.	Implemented already. Cannot be redefined.
GET_ID	Returns the ID of the current document.	Implemented already. Cannot be redefined.
GET_REPOSITORY	Returns an object of class CL_DM_DOCUMENTS. This object represents the repository.	Implemented already. Cannot be redefined.
CHECK_REPOSITORY_PROPERTY	Checks a repository attribute.	Implemented already. Cannot be redefined.

You also implement those GDMA interfaces that you require. You can find the documentation about GDMA interfaces in the section Using the Generic Document Management API (GDMA).

If you only require simple document services, you can inherit from the class CL_DM_SIMPLE_DOCUMENT. This class already contains the most important interfaces.

If you want to study an example implementation, you can look at the class CL_DM_TEST_DOCUMENT in the package SGDMA_TEST. This class implements the minimum functions.

Note for the connection to GSP:

If you are implementing a GDMA Service Provider that is to be called by the GSP, you *must* implement the following GDMA interfaces, since the GSP requires and calls these:

- IF_DM_DOCUMENT
- IF_DM_VARIANT (optional)
- IF_DM_VERSION
- IF_DM_LOCK
- IF_DM_ACCESS
- IF_DM_READ_CONTENT
- IF_DM_WRITE_CONTENT
- IF_DM_READ_URL
- IF_DM_WRITE_URL
- IF_DM_READ_FILE
- IF_DM_WRITE_FILE
- IF_DM_PROPERTIES
- IF_DM_COMPONENT

4.3 Cross-Document Functions

Declare a class that inherits from CL_DM_UNBOUND. This class is responsible for searching for documents, as well as for the authorization check, which does not refer to a particular document, rather to the repository.

The class CL_DM_UNBOUND has the following methods, which you must implement:

Method Name	**Explanation**	**Implementation**
CONNECT	Importing REP_IMPL. The method is called when the object is generated. You receive the interface reference to IF_DM_REP_IMPL. You can store this in a class attribute.	To be implemented

You must implement the interface IF_DM_QUERY and the interface IF_DM_ACCESS.

We recommend that you implement the interface IF_DM_ACCESS twice, to conduct the following two authorization checks:

- Checking t he access to a document. You can find a list of the activities that can be checked on page 24.
- Checking the access to the repository (for example, activities *Find* and *Create*). You can find a list of the activities that can be checked on page 25.

4.3.1 SAP Class

If you do not require an authorization check for the search, you can use the class CL_DM_FULL_REP_ACCESS. This is fully implemented, and can be registered in the registry directly.