SAP
Records Management

# Calling Records API

Developer Documentation

May 10, 2004

# Contents

# 1 Introduction

To make the best use of this document, you require a good working knowledge of the technical terminology and architecture of the Records Management Framework. For more information, see the Records Management reference documentation for developers.

The Records API provides you with methods for performing operations on records in the background. You can execute almost all activities that are provided for the user in the Records Browser.

Before you start working with the Records API, we recommend that you study the program SRM_RECORD_API_HOWTO. The following documentation refers to this program. Note: You can execute this program only if the values of the constants exist in the system. If you want to execute the program, you must adjust the constants in the program.

As well as the Records API, you can also use the BAPIs of the RECORD business object. These BAPIs are based on the Records API and are documented online in the system.

# 2 Steps for Instantiating the Records API

**Initialization:** You get a root handle for the Records Management Framework and a service object. You need to perform this step only if you are not in the Records Management context and the framework has not yet been instantiated.

For an example implementation, see the `initialize` subroutine in the SRM_RECORD_API_HOWTO program.

**Creating a POID:** When you create a new record, you need a model POID. You use the service object method IF_SRM_SRM_CLIENT_SERVICE~POID_CREATE_MODEL( ) to create this POID. For all other activities, you require an instance POID of the record. You use the IF_SRM_SRM_CLIENT_SERVICE~POID_GET_INSTANCE( ) method to create this instance POID.

For an example implementation, see the `createRecordModelPoid` or `createRecordInstancePoid` subroutines in the SRM_RECORD_API_HOWTO program.

**Creating a POID for a Record Element:** You need to perform this step only if you want to perform an operation on an element in the record. As an example, the SRM_RECORD_API_HOWTO program creates the POID of a note.

For an example implementation, see the `createNoteInstancePoid` subroutine.

**Instantiating the Records API:** In the service object, you can use the GET_SP_CONNECTION( ) method to request the back-end object of the POID. If you want to create a new record, send a model POID as an import parameter; in other cases, send an instance POID. The central interface of the Records API is IF_SRM_SP_RECORD. You get this interface by casting the back-end object.

For an example implementation, see the SRM_RECORD_API_HOWTO program.

# 3 Functions of the Records API

The Records API contains three interfaces:

## 3.1 IF_SRM_SP_RECORD

This interface includes methods for operations on a record. You use the method described above to instantiate an object.

For each operation on a record, you must first call the OPEN( ) method. This method loads the record. You use the FOR_UPDATE parameter to set a flag that indicates whether the record is called in change mode or not. You can then perform record operations. Afterwards, you must always call the SAVE( ) and CLOSE( ) methods and execute a COMMIT WORK.

The following functions are available:

### Creating a Record

CREATE( ): Creates a new record based on the element type identified by the model POID.

You use the MODEL_DOCID parameter to specify the record model on which the record is based. Enter the document ID of the record model. This ID comprises the document class (LOIO class of the content model) and the object ID (GUID). You use the *Information* activity of the record model to get the document ID. You must also set the record number (ID) and a description (DESCRIPTION). If you want to check whether the record model is unique, set the ID_CHECK_UNIQUE parameter.

For an example implementation, see the SRM_RECORD_API_HOWTO program. Note: The code is only inserted as a comment.

### Deleting a Record

DELETE( ): Deletes the record identified by the instance POID.

For an example implementation, see the SRM_RECORD_API_HOWTO program. Note: The code is only inserted as a comment.

### Inserting an Element in the Record

Several methods are available:

ELEMENT_ADD_BY_ANCHOR( ): The position at which the element is inserted into the record is determined by the ANCHOR import parameter. The anchor is an attribute that can be specified for a model node in the record model. The value that you specify for the ANCHOR parameter must be defined in the record model. In the record, the element is appended to precisely this model node.

ELEMENT_ADD_BY_PARENT_ANCHOR( ): Unlike the ELEMENT_ADD_BY_ANCHOR( ) method, this method uses the parent node to specify the position. This is necessary in those cases where subtrees are duplicated and an anchor ap pears at multiple positions in the record. You can use either the node ID (PARENT_NODE_ID) or a node attribute (PARENT_NODE_ATTRIBUTE) to identify the parent node.

ELEMENT_ADD_BY_MODELID( ): The position at which the element is inserted into the record is determined by the MODEL_ID import parameter. The model ID is the node ID of a model node in the record model. The specified ID must exist in the record model. In the record, the element is appended to the model node with this ID.

ELEMENT_ADD_BY_PARENT_MODELID( ): Unlike the ELEMENT_ADD_BY_MODELID( ) method, this method uses the parent node to specify the position. This is necessary in those cases where subtrees are duplicated and a model ID appears at multiple positions in the record. You can use either the node ID (PARENT_NODE_ID) or a node attribute (PARENT_NODE_ATTRIBUTE) to identify the parent node.

ELEMENT_ADD_FIRST_FIT( ): The position at which the element is inserted in the record is the first model node for which the element type of the element is stored in the record model.

ELEMENT_ADD_BY_REFERENCE( ): This method is available from WebAS Release 7.0. You can use this method to insert an element at a precise position in a list of elements. The position at which the element is inserted into the record is determined by the REFERENCE_ELEMENT_ID import parameter. This can be the ID of any element that has already been inserted. You can use the STACKED parameter to determine whether the new element is inserted before or after the reference element (IF_SRM=> TRUE if before the element; IF_SRM=>FALSE if after). You can use the ADD_AS_CHILD parameter to specify that the element is created as a node one level down. Note: It is up to you to make sure that the node matches the chosen position. If it does not, an exception is raised.

All methods have the following import parameters:

ELEMENT (interface reference to IF_SRM_SP_RECORD_ELEMENT):
You use the IF_SRM_SP_RECORD~ELEMENT_CREATE( ) method to get a reference to the IF_SRM_SP_RECORD_ELEMENT interface. This interfa ce includes methods that enable you to specify information for the new element. The following two steps are mandatory:

- Specify whether the new element is a structure node, instance node, or model node. To do this, use the IF_SRM_SP_RECORD_ELEMENT~TYPE_SET( ) method. You use one of the constants of the IF_SRM_SP_RECORD_ELEMENT interface, TYPE_FOLDER or TYPE_INSTANCE, as the value of

the TYPE import parameter.

- Specify the instance POID of the new element. You set the instance POID in the IF_SRM_SP_RECORD_ELEM_INSTANCE interface. You get a reference to this interface by casting the IF_SRM_SP_RECORD_ELEMENT interface.

  For an example implementation, see the `fillRecordElement` subroutine in the SRM_RECORD_API_HOWTO program.

STACKED:
Specify whether the element is inserted at the top or bottom of the list of elements for a model node. By default, this parameter is set to IF_SRM=>TRUE and the element is inserted at the top.

For an example implementation, see the SRM_RECORD_API_HOWTO program.

**Inserting an Element as a Subnode for an Existing Element**

A model node can appear more than once in a record. This can happen, for example, if complete subtrees are instantiated more than once. You can use the following methods to specify an exact position by identifying the model node and by identifying the previously instantiated parent node. This can be a structure node or an existing element.

ELEMENT_ADD_BY_PARENT_MODELID( ): You identify the model node by specifying the ID of the model node (MODEL_ID parameter). You can use either of two parameters to identify the parent node. Give one of the following two parameters a value:

- PARENT_NODE_ID; uses the node ID in the record to identify the parent node
- PARENT_NODE_ATTRIBUTE; uses node attributes to identify the parent node

The ELEMENT and STACKED parameters are used in the same way as the parameters for inserting an element described above.

ELEMENT_ADD_BY_PARENT_ANCHOR( ): You identify the model node by specifying the anchor of the model node (ANCHOR parameter). You can use either of the parameters PARENT_NODE_ID or PARENT_NODE_ATTRIBUTE to identify the parent node (see above). The ELEMENT and STACKED parameters are used in the same way as the parameters for inserting an element described above.

For an example implementation, see the SRM_RECORD_API_HOWTO program.

**Getting the Elements of the Record**

ELEMENT_GET_ALL( ): Gets all elements of the record in a list.

ELEMENT_GET_BY_ID( ): You send the node ID of a record element and get the element of this ID back.

ELEMENT_GET_BY_MODELID( ): You send the ID of a model node from the record model and you get all elements back that have been inserted at this model node in the record.

ELEMENT_GET_BY_TYPE( ): You send a node type (one of the three constants of the IF_SRM_SP_RECORD_ELEMENT interface: TYPE_FOLDER, TYPE_MODEL, or TYPE_INSTANCE). You get back all elements that have this node type in the record.

For an example implementation, see the SRM_RECORD_API_HOWTO program.

**Changing an Element of the Record**

ELEMENT_ATTRIBUTES_UPDATE( ): Changes the node attributes of an element.

ELEMENT_DESCR_UPDATE( ): Changes the description of the element node.

ELEMENT_ROLES_UPDATE( ): Changes the roles whose owners can view the element in the record hierarchy. In a dialog, a user can save roles in both the record model and the record. Roles in the record overwrite any roles that are saved in the record model. You can use this method to change the roles that are saved in the record.

ELEMENT_RELATIONS_UPDATE( ): Changes the relations of an element. A relation re fers to the relationship between the record and an element. In a dialog, a user can save relations only in the record model for a model node. You can use this method to set relations for an element directly in the record.

For all these methods, you specify a reference to the IF_SRM_SP_RECORD_ELEMENT interface. You get this reference as a return parameter of the method IF_SRM_SP_RECORD~ELEMENT_GET_* (see above) or IF_SRM_SP_RECORD~ELEMENT_CREATE( ). You can set the changes with the methods of the IF_SRM_SP_RECORD_ELEMENT interface.

For an example implementation, see the SRM_RECORD_API_HOWTO program.

**Deleting an Element from the Record**

ELEMENT_DELETE_BY_ID( ): Removes an element from the record. You use the node ID of the element in the record to identify it. If the element has subelements, then they are also deleted from the record.

ELEMENT_DELETE_BY_POID( ): Removes an element from the record. You use the POID of the element to identify it. If the element appears more than once in the record, every instance is deleted.

For an example implementation, see the SRM_RECORD_API_HOWTO program.

**Closing or Opening a Record**

FREEZE_RECORD( ): The record is closed and can no longer be changed.

UNFREEZE_RECORD( ): The record is opened and can be changed again.

IS_RECORD_FROZEN( ): Tells you whether the record has been closed.

**Getting the Record Log**

GET_RECORD_PROTOCOL( ): Gets all log entries for the record over a specified period.

**Extracting the Record Hierarchy (Iterative)**

ITERATOR_CREATE( ): Gets an interface reference to IF_SRM_SP_RECORD_ITERATOR. This interface includes methods for extracting child elements for a given node. The node ID in the record is used to identify the parent node. By default the root node (ID *ROOT*) is sent.

For an example implementation, see the SRM_RECORD_API_HOWTO program.

## 3.2 IF_SRM_SP_RECORD_CONTEXT

This interface includes methods that define properties for record operations. You get a reference to this interface by casting the IF_SRM_SP_RECORD interface.

The following functions are available:

ACCESS_CHECK_SET( ): Determines whether the authorization check for accessing the record is activated or deactivated. The check is active by default.

For an example implementation, see the SRM_RECORD_API_HOWTO program.

IM_UPDATE_TEST_SET( ): Determines wh ether the update is activated or deactivated. The update is deactivated by default.

RELATION_MASTER_SET( ): Internal use only

## 3.3 IF_SRM_SP_EXPERT

This interface is for internal use only.