Master Data Governance (MDG) Application Programming Interface (API) Guide

Applies to:

SAP MDG 9.1 and lower

Summary

This guide provides an overview of existing APIs for MDG and gives an overview of the Governance API and the Convenience API, explaining capabilities and behavior. It also contains sample code.

Author(s): Marcus Galleck, Dr. Jürgen Reibel

Company: SAP SE

Created on: 07 June 2016 Version 2

Author Bio

Marcus Galleck, SAP SE

Development Architect

Dr. Jürgen Reibel, SAP SE

Development Expert

Table of Contents

1 MDG Usable APIs

1.1 API Overview

The MDG framework provides several interfaces that are used by the delivered SAP MDG applications and also can be used by other external applications (like customer applications) in order to handle the governance process and the related entity data. These are the interfaces:

API	Interface	Implementing Class
Change Request API	IF_USMD_CREQUEST_API	CL_USMD_CREQUEST_API
External Model API	IF_USMD_MODEL_EXT	CL_USMD_MODEL_EXT
Governance API	IF_USMD_GOV_API	CL_USMD_GOV_API
Convenience API	IF_USMD_CONV_SOM_GOV_API	CL_USMD_CONV_SOM_GOV_API
Context API	IF_USMD_APP_CONTEXT	CL_USMD_APP_CONTEXT

***Table 1**: Relevant APIs*

1.2 Main Purposes

The different APIs have different purposes which are described in the following table:

API	Main Purpose
Change Request API	Handling of change requests and partly changing and reading entity data
External Model API	Reading entity data
Governance API	Handling of (several) change requests and entity data
Convenience API	As the Governance API but using a single change request
Context API	Store and provide process context data

***Table 2**: Main purposes of the relevant APIs*

Originally, only the Change Request API and the External Model API existed. The Change Request API provides the possibility to fully control the governance process and the life cycle of a change request. This includes the possibilities to create, save, activate or reject a change request, to modify its object list as well as to read and change entity data.

The External Model API was provided to enable extended possibilities to read entity data.

Since the Change Request API and the External Model API turned out to be quite complex to use from the caller perspective, it was decided to provide a new (Governance) API. The focus of the Governance API is, to provide a single API with full functional scope concerning the coverage of the governance process using workflows (change request handling) and the correlated handling of the entity data. Furthermore, this API was designed to stay as simple as possible by hiding several automatic steps. It still needs to fulfill all new requirements arising from the different MDG applications.

The Convenience API was developed in parallel to the Governance API. This API is on top of the Governance API with the focus of being even simpler to use. This improved simplicity is reached by reducing its scope to the handling of a single change request. Its main purpose is to be used for single object maintenance UIs where its usage is restricted not only to use a single change request but also to change a single object only as well.

Later it was discovered that lower architectural levels sometimes need access to some process relevant data. To enable this, the (Application) Context API was developed.

1.3 Availability and Functional Scopes

The existence of the different APIs and their functional scope depends on the MDG release. The following is valid independent of the underlying system being an ERP or SAP S/4 HANA On-Premise system.

In EhP5 only the Change Request API and the External Model API existed. Both APIs can still be used in the later MDG releases. Whereas the External Model API always was kept up to date with new developments, the development of the Change Request API was stopped in EhP6. However, with respect to change request handling the Change Request API offers some functions that are not covered by any other API. This includes the possibility to move objects between different change requests, to split one change request including to move objects to the new change request, and to create a change request using another change request as a template.

In EhP6 the Convenience API and the Governance API were introduced and their functional scopes were further extended from release to release. Most of these developments were done to fulfill new requirements coming from the later MDG applications.

The External Model API is fully up to date in each MDG release.

The following table provides a rough overview of the situation.

MDG Release	Description	Functional Scope
EhP 5	Change Request API	Programming interface for CR Processing. One Instance per change request. Administration of change request object list by consumer.
	External model API	Reading entity and hierarchy data.
EhP 6	Change Request API	No further development.
	Governance API	Programming interface for governance process. Multiple CR processing within one instance for one model. Limited use for models using editions.
	Convenience API	Programming interface for governance process optimized for single object processing and single object processing UIs. No support for models using editions.
	Context API	Programming interface consumed mainly by convenience API. Stores and calculates context data.
	External model API	Reading entity and hierarchy data.
MDG 6.1	Change Request API	No further development.
	Governance API	Support of foreign key entities.
	Convenience API	Support of foreign key entities.
	Context API	No further development.
	External model API	Reading entity and hierarchy data.

MDG 7.0	Change Request API	No further development.
	Governance API	Support of new edition management and parallel change requests
	Convenience API	Support of new edition management, parallel change requests and highlighting changes
	Context API	Support of new edition management and parallel change requests
	External model API	Support of new edition management and parallel change requests
MDG 8.0	Change Request API	No further development.
	Governance API	Hierarchy maintenance
	Convenience API	Hierarchy maintenance
	Context API	No further development.
	External model API	No further development.
MDG 9.0 and further releases	Change Request API	No further development.
	Governance API	No further development
	Convenience API	No further development.
	Context API	No further development.
	External model API	No further development.

***Table 3**: Functional Scopes of the relevant APIs in dependence of the MDG release.*

1.4 Architectural Perspective

From the architectural point of view the main part of the MDG framework is the Abstraction Layer. The External Model API has a direct access to this layer and can be used from all architectural layers including deeper ones like access classes and BADI implementations. The Change Request API also directly accesses the abstraction layer. The App Context API is besides these layers and can be consumed independently.

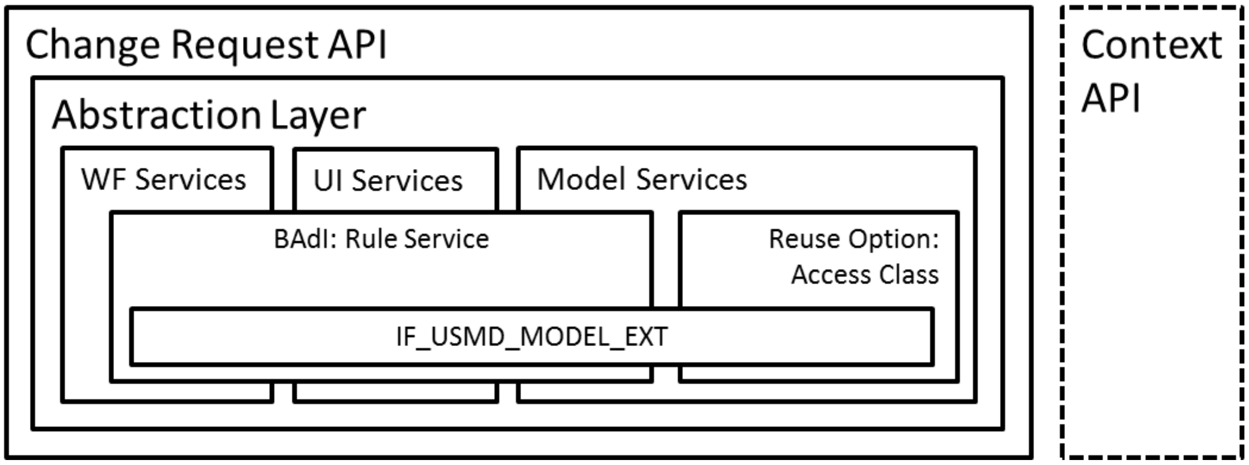

***Figure 1**: Architectural layers with respect to the Change Request API*

From the architectural point of view the Governance API is built on top of the abstraction layer and the Convenience API on top of the Governance API. Please notice that the Convenience API is the only API which somehow covers the APP Context API that is because it is the only one which automatically fills the application context with information.

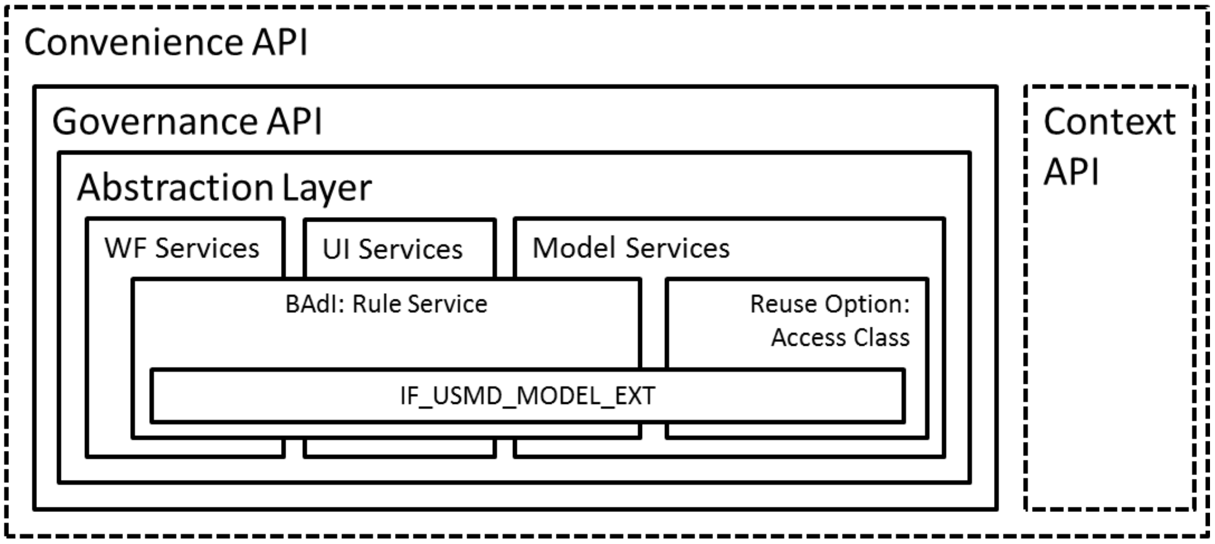

***Figure 2**: Architectural layers with respect to the Governance and Convenience APIs*

2 Limitations

When using the described APIs a number of limitations have to be considered which are explained in the following section. These limitations are true for one instance of the respective API within one logical unit of work (LUW).

2.1 Limitation with Respect to the Functional Scope

As already discussed above, the functional scopes of the different APIs depend on the respective MDG release.

The Change Request API was not developed further since MDG EhP6 and therefore some of the newer features are not supported there. As an example, it is not possible to process hierarchies. Regarding the control of the governance process, it offers some quite specific process interactions (like explicitly creating, rejecting or activating a change request) but on the other hand delegates the responsibility for handling the whole process completely to the caller. This includes that the caller is responsible that the changed data and the change request object list always are in a consistent state with respect to each other (e.g. before checks are performed or the changes are saved).

The functional scopes of the Convenience API and the Governance API were extended from release to release. They do not offer the same explicit process interaction possibilities as the Change Request API, but on the other hand they free the caller from some of the responsibilities to perform a consistent governance process. They do this by strictly following the governance process of the selected change request type and by triggering the respective workflow.

As already mentioned the scope of the External Model API is restricted to read data and it does not provide any possibility to change data.

The purpose of the context API is to store some governance process relevant data and to allow callers to request this information at a later point in time and at a completely different place in the call stack. However, the Context API can only provide the data which had been filled in beforehand. This data filling is only provided by the Convenience API automatically. In all other cases the caller application is responsible to fill in the correct and needed information before it is requested the first time. The Convenience API fills the Context API as completely as possible. This means that by instantiating the convenience API the context is only filled with the model name. After providing a change request type for creating an entity or providing the change request ID for changing an entity, the context is enriched by change request type, the business process, and the change request ID. However, because of the different possible usages of the Convenience API itself there is no guarantee for an always filled Context API. It must be checked from case to case if the requested information is available.

2.2 Limitations with Respect to the Data Model

One API instance can only handle one dedicated data model. These is true for all the APIs mentioned.

2.3 Limitations with Respect to the Number of Change Requests

One instance of the Convenience API and one instance of the Change Request API can handle only one change request at a time. However, within the runtime of an application they can handle multiple change requests sequentially when the LUWs are changed before changing a change request. This means, before changing from one change request to another one, the internal buffers must be refreshed, the context (when used) must be updated, the data changes must be saved, and a commit work must be triggered.

2.4 Limitations with Respect to the Parallel Usage of the APIs

The Convenience API, the Governance API and the Change Request API are not stateless and not independent on each other. **It is not allowed, to use two of these APIs within the same LUW.** In the case of the Convenience API and the Governance API there is even a protection integrated, that these two APIs cannot be instantiated within the same LUW (at least not for the same data model).

The External Model API and the Context API are independent on the other APIs and can always be used in parallel to all other ones.

2.5 Limitations with Respect to the Caller

From the architectural point of view not all the APIs can be called from everywhere. The Convenience API, the Governance API and the Change Request API are designed to be called from external applications (such as UI layers and services). However, **it is not allowed to call these interfaces from architecturally deeper layers like access classes or any BADI implementations**. This error happens sometimes at customer implementations and then can lead to complex error situations. A lot of effort is then needed to find out this root cause.

The External Model API and the Context API have no such caller restrictions and especially for the External Model API the calls from access classes and BADI implementations belong to the most important use cases. The only exception of this is that the External Model API may not be called out of the READ methods of access classes because this could lead to undesired recursions within the call stack and resulting short dumps.

2.6 Limitations with Respect to Business Functions

To be able to use MDG at all, the MDG foundation business function MDG_FOUNDATION for EhP5 must be switched on. This is already sufficient for using the Change Request API, the External Model API and the Context API.

For being able to use the Convenience API and the Governance API in addition the business function MDG_FOUNDATION_2 for EHP6 must be switched on in addition.

2.7 Other Limitations

As a prerequisite for the APIs to work reliably, the overall MDG application must be configured completely and correctly. This includes the required customizing settings like consistent active data models, correctly specified change request types and workflow settings. Furthermore, within external extensions (For example, customer BADI implementations) the mentioned APIs must be used considering all the limitations mentioned above. The recommendation is not to use the IF_USMD_MODEL at all.

3 Interfaces

In this chapter, only an idea of the interfaces should be given. For more detailed information please refer to the system documentation (interface-, class- and method documentation).

3.1 Change Request API

The Interface of the Change Request API is not explained here as it is replaced by the Governance/Convenience API.

3.2 External Model API

The External Model API is used to read entity data and hierarchy data (in case the model supports hierarchies) and creating data references for entities. The External Model API can be instantiated directly using the static method GET_INSTANCE of the class CL_USMD_MODEL_EXT or it is given in some other interfaces. The External Model API can be instantiated with a valid data model. Exceptionally the initial Model (I_USMD_MODEL = SPACE) is a valid model for instantiating the External Model API. In case the initial model is used only change request data can be read.

3.2.1 IF_USMD_MODEL_EXT

The methods of this interface can be used to read entity data and change request data, to create data references for further processing and to get an instance for hierarchy data access.

3.2.2 IF_USMD_MODEL_CR_EXT

The interface contains methods to read change request data (e.g. the object list of a change request).

3.3 Governance API

The governance API has a static factory method, GET_INSTANCE, to provide an instance of the requested API for a specific data model. The data model is mandatory for the API. To request the API several times for one data model, the same instance of the API is provided by the GET_INSTANCE method. GET_INSTANCE allows applications to inherit from this class and provide an own implementation, which can provide additional methods for your convenience (for example, providing a method to read a business partner with the structure BUT000 instead of reading the entity type BP_HEADER with the ABAP statement *REF TO DATA* as the export parameter).

The governance API CL_USMD_GOV_API (IF_USMD_GOV_API) includes other interfaces for better clarity, since there are many methods involved. Nearly all methods of the governance API are documented in the system. The functionality, the requirements and the expected results of a particular method are briefly described in the method documentation.

The governance API is consumed by the convenience API and the data load process.

3.3.1 IF_USMD_GOV_API

The interface IF_USMD_GOV_API combines all necessary sub-interfaces to support the governance process. The purpose of the sub-interfaces is to categorize the methods for their use. There is one event defined for this interface that is raised as soon as the data has been changed due to derivations or data enrichments.

3.3.2 IF_USMD_GOV_API_CR_DATA

The methods of this interface can be used for change request header data, change request notes, change request attachments, and target systems.

3.3.3 IF_USMD_GOV_API_ENTITY

The methods of this interface can be used for data manipulation based on the governance process. There, methods can be used for locking an entity, reading and writing entity data, etc.

The attributes of this interface are constants for creating data references of different kinds.

3.3.4 IF_USMD_GOV_API_PROCESS

The methods of this interface can be used to check the data of a change request and to create and forward the workflow.

3.3.5 IF_USMD_GOV_API_SERVICES

The methods of this interface can be used to retrieve change requests by a specific entity as well as for step type determination for a specific change request, and for the determination of permitted changes for a change request in process.

3.3.6 IF_USMD_GOV_API_TRANS

This interface contains two methods that can be used to save the change requests and the corresponding inactive entity data, and to refresh the master data governance buffer (change request data and entity data).

3.3.7 IF_USMD_GOV_API_CR_ACTION

This interface contains one method for the deletion of a change request in draft mode.

3.4 Convenience API

The Convenience API has a static factory method, GET_INSTANCE. For instantiation, it behaves like mentioned in the Governance API instantiation.

The Convenience API CL_USMD_CONV_SOM_GOV_API (IF_USMD_CONV_SOM_GOV_API) consumes the functions and methods of the Governance API. It is designed for easy and convenient consumption of the governance process within a specific object application. The application has to take care of the transactional behavior and application-specific data.

One important method is the SET_ENVIRONMENT method. As change requests might be created implicitly because there is no explicit method for creating a change request, the change request ID or at least the change request type needs to be set if you want to change an object for which no change request exists. For example, the creation of a change request is triggered by locking the main entity of a data model and by retrieving a temporary number for the main entity. Before an entity can be locked and a change request can be created, the environment has to be set. If you want to create a new change request, the change request type has to be defined.

Just like the governance API, the convenience API also includes other interfaces for better clarity.

3.4.1 IF_USMD_CONV_SOM_GOV_API

The interface IF_USMD_CONV_SOM_GOV_API combines all necessary sub-interfaces to support the convenient governance process. The purpose of the sub-interfaces is to categorize the methods for their use. There are three events defined for this interface that are raised in the following situations:

- data has been changed due to derivations or data enrichments
- data was saved
- the key of an entity has changed during activation

The methods of this interface can be used for message handling, setting of the environment, and for determining the processing details of an entity.

3.4.2 IF_USMD_CONV_SOM_GOV_CR

The methods of the interface IF_USMD_CONV_SOM_GOV_CR handle change request header data as well as change request notes, attachments, and target systems. With the method RETRIEVE_CREQUESTS_BY_ENTITY you can determine whether an entity is in process and which change request is used for processing. Methods relevant to the workflow are also available, for example the method SET_ACTION. It can be used to provide the action result of a workflow action to finalize processing, to approve, or to reject.

3.4.3 IF_USMD_CONV_SOM_GOV_ENTITY

The methods of interface IF_USMD_CONV_SOM_GOV_ENTITY handle entity data. Entity data can be enqueued. If you have not already set an existing change request ID as environment, enqueueing an entity or acquiring a new temporary entity key creates a change request. Otherwise a change request type needs to be provided. Data can be retrieved, written and checked. SAP also provides a method for field property determination.

3.4.4 IF_USMD_CONV_SOM_GOV_TRANS

The two methods of interface IF_USMD_CONV_SOM_GOV_TRANS are relevant to transactional handling. This interface contains the methods SAVE and REFRESH_BUFFERS.

3.4.5 IF_USMD_CONV_SOM_GOV_CR_ACTION

This interface IF_USMD_CONV_SOM_GOV_CR_ACTION contains only one method for the deletion of a change request in draft mode (no workflow has been started so far for the relevant change request).

3.5 CL_USMD_APP_CONTEXT

The Application Context can be used to get the current MDG context e.g. the currently used data model, current used process and current change request meta data. Application Context method GET_CONTEXT is used to get an instance of the context API interface IF_USMD_APP_CONTEXT. This instance can be used to read the relevant context information. Attention: The context contains only the data that has been filled in beforehand.

4 Data Handling

4.1 Introduction to the MDG Master Data Handling

In a master data governance process, there are different sources of entity data. On the one side, there are the active data. These are the valid data in the system, that is, the data which are accessed from all applications and business processes outside of MDG.

There are also the entities which are changed in a MDG governance process using a change request which are kept separately from the active data. This data is called inactive data or staging data. When the entity changes of a governance process are finally approved, the inactive data related to the respective change request are transformed into the corresponding active data. This process is called change request activation.

The MDG entity types can be divided into flex entity types and reuse entity types. The flex entity types have no own data storage for their active data but their active and their inactive data are stored within the MDG staging tables. All read and write accesses to flex entity data are performed the the MDG abstraction layer and the subsequent MDF layer.

Reuse entity types have their own data storage for their active data. The read and write accesses to this data is delegated from the MDG abstraction layer to respective access classes. Their inactive data is stored in the MDG staging tables as in the case of the flex entity types. In addition to the inactive data the reuse entity types also have the snapshot data which are a copied from the active data and which are created at the point in time when a reuse entity is inserted into a change request object list. As in the case of the inactive data, the snapshot data are stored in the MDG staging tables as well. The accesses to this data always happens from the abstraction layer using the MDF layer.

4.2 Reading Entity Data

The process of reading entity data is schematically shown in the following picture:

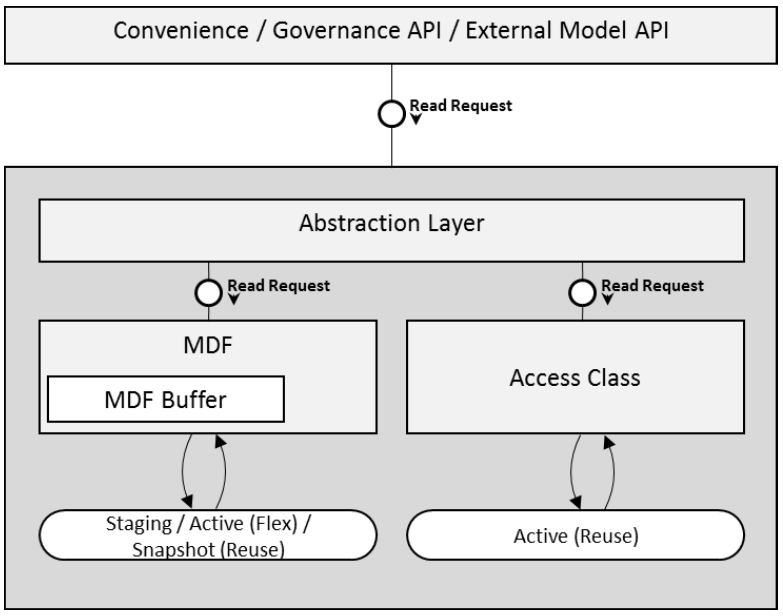

***Figure 3**: Architectural layers with respect to the reading of entity data*

Entity data can be read using the Convenience API, the Governance API, and the External Model API as well. In all cases, the read request is delegated to the MDG abstraction layer, which itself forwards the requests to the MDF layer and/or the access classes and returns the combined results to the respective caller.

Data that is changed within a LUW which have not yet been saved, are buffered within the MDF buffer which is located within the MDF layer. By default, read requests obtain the combined result of stored data adjusted by the content of the MDF buffer.

In general, the obtained result of a read request depends on the request parameters and the existence of respective data in the different storage locations and within the MDF buffer. The available request parameters depend on the API used and control which data will be read (For example, active versus inactive entity data, considering the assignment of inactive data to certain change requests, key selections, attribute selections, and editions).

The External Model API offers the widest variety on how a read request can be specified but on the other hand it is too complex to be explained here in detail. As already described above, the Governance API offers a simpler interface and therefore will be explained here as an example in some more detail. The main method for reading entity data with the Governance API is IF_USMD_GOV_API_ENTITY~READ_ENTITY. This method has the following importing parameters for controlling the read request:

1. IV_ENTITY_NAME: The entity type for which data should be read.
2. IF_ACTIVE_DATA: Specification if active or inactive data should be read.
3. IV_CREQUEST_ID: Since the Governance API supports the handling of several change requests, here the relevant change request for reading inactive data must be specified. Without specifying a change request, active data are read even if the parameter IF_ACTIVE_DATA is specified to read inactive data.
4. IT_KEY: With this table the keys of the entities to be read are specified. The keys must fit to the specified entity type. If no keys are specified no data are read at all.
5. IV_EDITION: In case of edition dependent entity types with this parameter the edition to be read is specified. If there is no edition supplied, but a change request is provided, the edition will be taken from there. If both is not provided, the data are read edition independently.

The following matrix describes the behavior using a data example:

Data storage / Read method parameters	Only active data available	Active data and inactive data in change request 4711	Only inactive data in change request 4711	Active data and inactive data in change request 0815	Only inactive data in change request 0815
IV_CREQEUST_ID = 4711; IF_ACTIVE_DATA = 'X'	Active data is read	Active data is read	No data is read	Active data is read	No data is read
IV_CREQEUST_ID = 4711; IF_ACTIVE_DATA = SPACE	Active data is read	Inactive data is read	Inactive data is read	Active data is read	No data is read

***Table 4**: Reading result depending on the existing data and the values of the importing parameters IF_ACTIVE_DATA and IV_CREQUEST_ID.*

4.3 Changing Entity Data

Entity data can be changed by writing processes or by derivations. In both cases during the change process the changed data are located in different locations until they possibly end up as active data. This process is explained in the following picture.

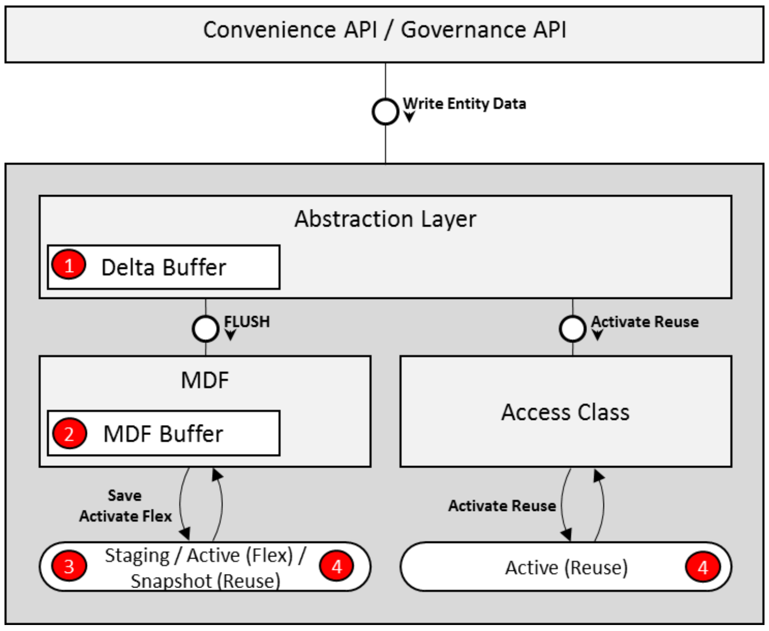

***Figure 4**: Architectural layers with respect to the changing of entity data*

Entity data can be written by the Change Request API, the Governance API or the Convenience API. In all these cases the write request is delegated to the MDG abstraction layer. The abstraction layer contains its own temporary Delta Buffer. It is important to know, that as long as changes are located in this buffer, they are not visible at any read accesses. To make them available to read accesses they must be transferred at first from the Delta Buffer into the MDF buffer, a process which is called flush. During such a flush, also some derivations are called which might further change the data.

To handle the different buffers and different data changing scenarios, the abstraction layer offers different write modes:

1. Direct Mode: With this write mode the changed data are directly written into the MDF buffer. The delta buffer is not involved at all and a flush is not taking place.
2. Direct Mode with Derivations: In this write mode the changed data are written into the Delta Buffer first and then the whole content of this buffer (including changes being collected beforehand) is transferred to the MDF buffer (flushed).
3. Collect Mode: In this write mode the changed data are collected in the Delta Buffer. The MDF buffer is not involved and a flush is not taking place.

A flush can be triggered from the applications at any point in time by calling explicitly the respective flush method. In addition to this a flush is normally automatically triggered when any read access happens or a write access with the respective write mode. This must be considered when evaluating the result of any reading data access. Especially by using the External Model API the flush can be suppressed by setting the corresponding importing parameter of the method IF_USMD_MODEL_EXT~READ_CHAR_VALUE. So, in any case after the flush all changed data are located in the MDF buffer, they might have be further changed by derivations and they are considered in read accesses.

When a change request is saved or submitted, the changed data are written as inactive data from the MDF buffer into the MDF staging tables. When the change request later on is approved and activated, these inactive data is converted into active data and end up either in the staging tables (flex entity types) or in the reuse tables (reuse entity types).

The MDG staging tables can be accessed with the report USMD_DATA_MODEL using transaction SE38.

4.3.1 Derivations

Derivations can be used to change entity data automatically without user interaction. The MDG framework offers several possibilities when, where and how derivations can be implemented. The most important ones are mentioned in the following:

1. Standard Derivations: The standard derivations can be implemented as BADI implementations of the BADI USMD_RULE_SERVICE in the method DERIVE_ENTITY of the BADI interface IF_EX_USMD_RULE_SERVICE or as BRF+ rules. The execution of these derivations is triggered by the abstraction layer method IF_USMD_RULE_SERVICE~DERIVE_ENTITY. This method is called at any write operation from the Governance API or the Change Request API and in addition by some application specific spots.
2. Derivations in Access Classes: The access class interface IF_USMD_PP_ACCESS provides the method DERIVE_DATA which can be implemented with derivations. This method is called per involved access class during a flush and gets the current content of the Delta Buffer to find out which data changes had been collected since the last flush was process.
3. Cross entity derivations: Those derivations can be implemented for the BADI USMD_RULE_SERVICE in the method DERIVE of the BADI interface IF_EX_USMD_RULE_SERVICE2. This method is also called during a flush and also gets the current content of the Delta Buffer in order to find out which data changes had been collected since the last flush was process. It is called after the derivations of the access classes and in addition obtains the information which data had been derived there.

All the derivation methods get the currently changed data (i.e. all collected data of the delta buffer since the last flush happened) imported using suitable importing parameters. If some additional entity data must be read within the derivation methods, the External Model Interface must be used (never the Convenience or Governance API). To avoid unnecessary flushes (and derivations), the NO_FLUSH parameter should be set to ‘X’.

In case of parallel change requests, it is possible that derived entity types are not in the scope of the change request type or that a derived entity is already interlocked in another change request. In such a case the derived data which cannot be added to the change list are filtered out. Only those derivations are taken over into the MDF buffer, that are in scope and can be interlocked. The removed entities are listed in a warning message. If the derived data is essential for data consistency a corresponding check should take care of this. The derivation provides no error message in such a case.

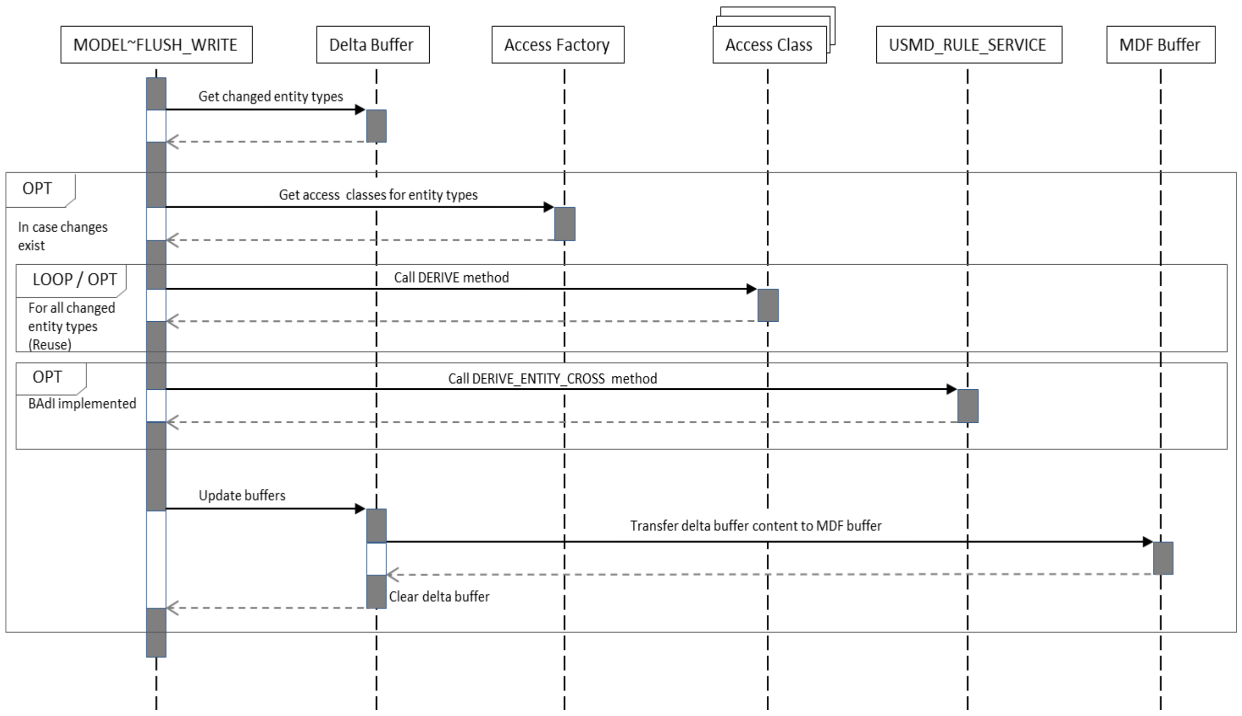

Figure 5: *Sequence diagram of the flush process and the called derivations.*

4.3.2 Writing Data with Convenience/Governance API

Since the Convenience API forwards all writing requests directly to the Governance API, the same mechanism is true for both APIs. In order that entity data can successfully be written in a Single Object Maintenance Application using the Convenience or Governance API, several prerequisites must be fulfilled:

1. The entity must not be interlocked by an already existing open change request. That means that the main object of the entity must not be part of the object list of an open classical change request and that the entity itself must not be contained in the change list of an open parallel change request.
2. The main object of the entity must successfully be enqueued.
3. The user has the authorization to create (if it is a new entity) or change (if it is an existing entity) the entity data.
4. At least one suitable change request type must exist in the customizing. In the case of parallel change request types the relevant entity type must be in the scope.

If these prerequisites are fulfilled, the Governance API automatically performs the following tasks:

1. If not yet done, the API creates a suitable change request. If only one suitable change request type exists in the customizing, this one is automatically used. Otherwise the user is asked to select the desired change request type. For edition-dependent entities the same mechanism is used to determine the edition to be used.
2. If not yet done, the main object of the entity is automatically added to the object list of the change request.
3. If not yet done, in the case of a parallel change request the entity is automatically added to the change list of the change request.
4. In case of a reuse entity type the relevant snapshot is created.
5. The data changes are written into the abstraction layer.

The main method for writing entity data with the Governance API is IF_USMD_GOV_API_ENTITY~WRITE_ENTITY. The interface of this method has the following parameters:

1. IV_CREQUEST_ID: The change request related to the data changes
2. IV_ENTITY_NAME: The entity type related to the data changes
3. IT_DATA: A table for specifying the changed data records. The structure of this table is forwarded to the called standard derivations and therefore only the attributes which are provided here are available in these derivations at runtime. Due to this reason, it is recommended to provide the full data structure of the respective entity here, although probably only a few of the data fields are changed.
4. IT_ATTRIBUTE (optional): This parameter is available in newer releases. In case only some of the attributes of the provided structure are changed, it is recommended to provide the names of the changed attributes in this table. Only if this is done, in the called standard derivations the information which fields have been changed is available.

The Convenience and the Governance API always write with the write mode collect. During the runtime of such a method execution it is guaranteed that a flush is not triggered. Several calls of the write method in a loop can therefore be used to really collect some data within the Delta Buffer.

4.3.3 Writing Data with the Change Request API

When writing entity data with the Change Request API, it only checks the authorization for the provided entity and the existence of a corresponding main object entry in the change request object list. The change request API does not offer an automatism to write the correct object list entries by writing entity data. This must be done by the API caller itself. When any entity data are written by the Change Request API, the standard derivations are called automatically and a flush is directly executed. That means that all the derivations mentioned above are performed.

4.4 Authorization

4.4.1 Entity Authorization

While reading and writing entity data to a change request, an authority check is performed. To read entity data, the authorization check is done for the display authorization (AUTH_ACT = 3). To write entity data, the authorization check action depends on whether active data exists. If the entity data exists only as inactive data (no active data is available), the authorization check verifies, if a user is authorized to create the entity (AUTH_ACT = 1). If active data is available, the authorization check verifies, if a user is authorized to change the entity (AUTH_ACT = 2).

4.4.2 Change Request Authorization

During the creation of a change request, the authorization action create (AUTH_ACT = 1) is necessary for the relevant change request type.

To create a workflow, the CREATE (AUTH_ACT = 1) authorization for a specific change request type is required. To complete a workflow step, the CHANGE (AUTH_ACT = 2) authorization for the change request type is required.

To change the object list of the change request, it is not necessary to check the change request authorization.

5 Simple Guide for Implementation

5.1 Sequence Diagram for Convenience API Usage

The following figure focuses on the convenience API and its usage. The governance API consumes the relevant methods from the abstraction layer.

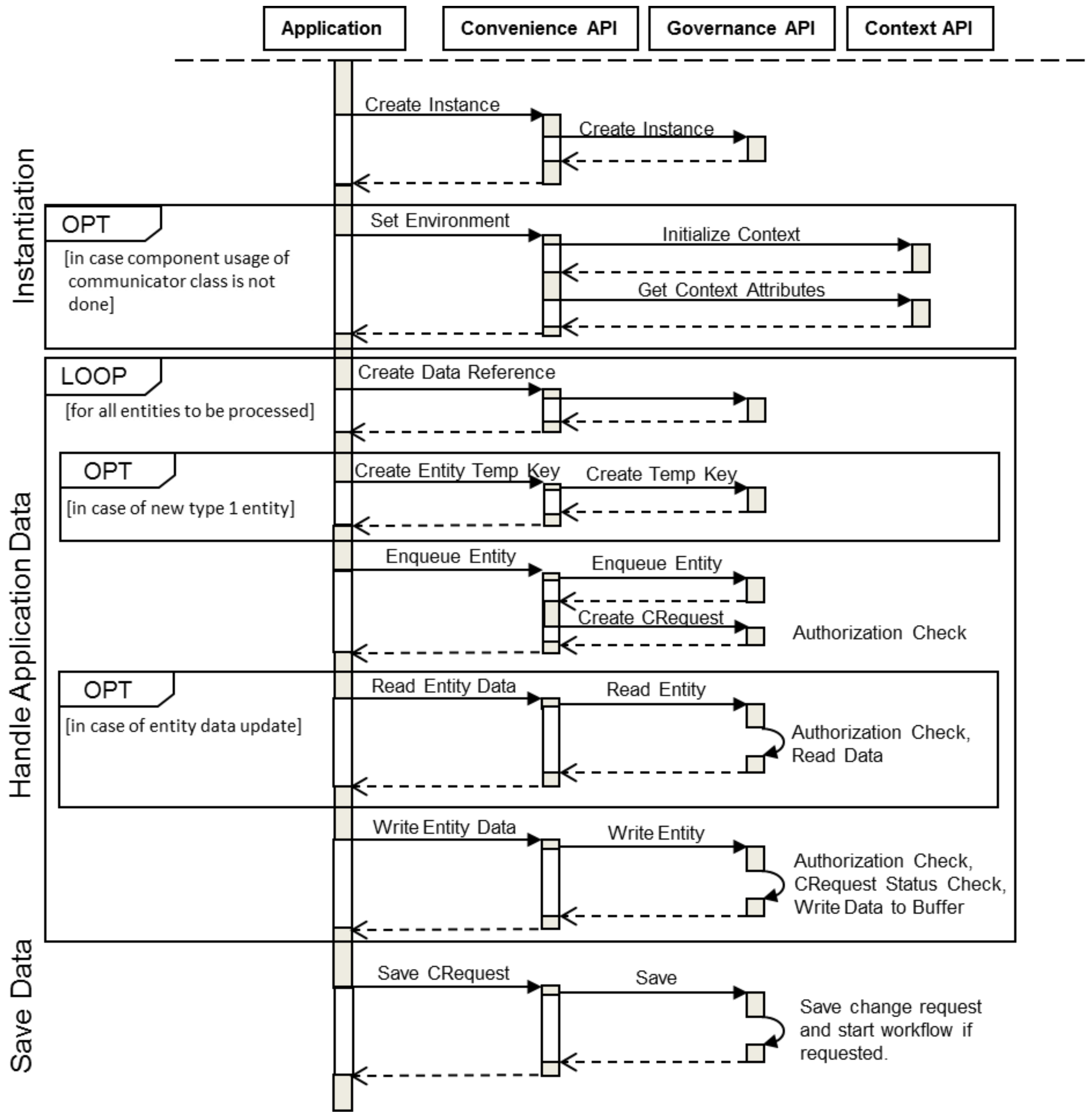

Figure 6: *Sequence diagram of change request handling.*

5.2 Code Example – Create Airline (with governance API)

```
*&---------------------------------------------------------------------*
*& Report  zcreatecarr_sf
*&---------------------------------------------------------------------*
*& This example creates a new change request. Along with the change
*& request, a new airline (carrier) is created.
*&
*& The exceptions raised by the governance API contain the error
*& messages and some other attributes. Therefore, it is possible to react
*& to the errors raised by the governance API.
*&---------------------------------------------------------------------*

REPORT zcreatecarr_sf.

DATA:
lo_gov_api        TYPE REF TO if_usmd_gov_api,
lv_crequest_id    TYPE usmd_crequest, "Change Request ID

lr_carr_key_str   TYPE REF TO data,  "Entity Carrier - key structure
lr_carr_key_tab   TYPE REF TO data,  "Entity Carrier - key table
lr_carr_data_str  TYPE REF TO data,  "Entity Carrier - data structure
lr_carr_data_tab  TYPE REF TO data,  "Entity Carrier - data table

ls_entity         TYPE usmd_gov_api_s_ent_tabl,
lt_entity         TYPE usmd_gov_api_ts_ent_tabl,

lt_messages       TYPE usmd_t_message.

FIELD-SYMBOLS:
<ls_carr_key>     TYPE any,
<lt_carr_key>     TYPE ANY TABLE,
<ls_carr_data>    TYPE any,
<lt_carr_data>    TYPE ANY TABLE,
<value>           TYPE any.

"1st: Create an instance of the governance API
TRY.
    lo_gov_api = cl_usmd_gov_api=>get_instance( iv_model_name = 'SF' ).
  CATCH cx_usmd_gov_api.
    EXIT.
ENDTRY.

"2nd: Create all the data references needed to maintain the carrier entity
"Create a data reference of the key structure / table of entity CARR (Carrier)
lo_gov_api->create_data_reference(

       EXPORTING iv_entity_name = 'CARR'
                 iv_struct      = lo_gov_api->gc_struct_key
      IMPORTING er_structure    = lr_carr_key_str
                 er_table       = lr_carr_key_tab ).

"Create a data reference of the key and attribute structure / table of

"entity CARR (Carrier)
lo_gov_api->create_data_reference(

       EXPORTING iv_entity_name = 'CARR'
                 iv_struct      = lo_gov_api->gc_struct_key_attr
      IMPORTING er_structure    = lr_carr_data_str
                 er_table       = lr_carr_data_tab ).


"Assign the created data references for carrier key and carrier data
```

```
"to field symbols
ASSIGN lr_carr_key_str->* TO <ls_carr_key>.
ASSIGN lr_carr_key_tab->* TO <lt_carr_key>.

ASSIGN lr_carr_data_str->* TO <ls_carr_data>.
ASSIGN lr_carr_data_tab->* TO <lt_carr_data>.

"3rd: Fill the key and data structure with values to create a new carrier
"The entity CARR only has key field CARR. The new carrier ID should be 'YZ'
ASSIGN COMPONENT 'CARR' OF STRUCTURE <ls_carr_key> TO <value>.
IF sy-subrc = 0.
  <value> = 'YZ'.
  INSERT <ls_carr_key> INTO TABLE <lt_carr_key>.
ELSE.
  EXIT.
  "Tough luck - unfortunately, this field name is not part of the key structure
ENDIF.

"4th: Create a new change request using change request type and a

"description (required)
TRY.
    lv_crequest_id = lo_gov_api->create_crequest(

                     iv_crequest_type = 'SFC01'
                     iv_description   = 'Create new Carrier YZ' ).


  CATCH cx_usmd_gov_api.
    "Something went wrong while creating the change request (e.g. model blocked

    "or change request type unknown).
    EXIT.
ENDTRY.

"5th: Before making changes to an object, the object needs to be enqueued

"even if this is a creation scenario
TRY.
    lo_gov_api->enqueue_entity( EXPORTING iv_crequest_id = lv_crequest_id
                                          iv_entity_name = 'CARR'
                                          it_data        = <lt_carr_key> ).
  CATCH cx_usmd_gov_api_entity_lock cx_usmd_gov_api.
    EXIT.
    "Tough luck -

    "something went wrong while enqueueing the entity (it could be a
    "technical reason, or maybe the carrier is already interlocked?!
ENDTRY.

"6th: Provide some entity attributes (complete data structure)
MOVE-CORRESPONDING <ls_carr_key> TO <ls_carr_data>.

ASSIGN COMPONENT 'CARRNAME' OF STRUCTURE <ls_carr_data> TO <value>.
<value> = 'Fantasy Flight Airlines'.

ASSIGN COMPONENT 'CURRCODE' OF STRUCTURE <ls_carr_data> TO <value>.
<value> = 'USD'.

ASSIGN COMPONENT 'URL' OF STRUCTURE <ls_carr_data> TO <value>.
<value> = 'http://www.fantasyflight.com'.

INSERT <ls_carr_data> INTO TABLE <lt_carr_data>.

"7th: Write the entity data to the change request
TRY.
```

```
    lo_gov_api->write_entity( EXPORTING iv_crequest_id = lv_crequest_id
                                        iv_entity_name = 'CARR'
                                        it_data        = <lt_carr_data> ).
  CATCH cx_usmd_gov_api_entity_write.
    EXIT.
    "Tough luck - might be that you have no authorization, or the entity is
    "not enqueued or cannot be added to the object list of the change

    "request
ENDTRY.

"8th: optionally, the entity data is read again... just to make sure everything
"went right.
TRY.
    lo_gov_api->read_entity( EXPORTING iv_crequest_id   = lv_crequest_id
                                       iv_entity_name   = 'CARR'
                                       it_key           = <lt_carr_key>
                             IMPORTING et_data           = <lt_carr_data> ).
  CATCH cx_usmd_gov_api_core_error cx_usmd_gov_api.
    EXIT.
    "Adequate Exception handling
ENDTRY.

"9th: The complete change request should be checked before it is saved
TRY.
    lo_gov_api->check_crequest_data( iv_crequest_id = lv_crequest_id ).
    "Collect the entities to be checked
    ls_entity-entity = 'CARR'.
    ls_entity-tabl   = lr_carr_key_tab.
    INSERT ls_entity INTO TABLE lt_entity.
    "check the entity
    lo_gov_api->check_complete_data(

                          EXPORTING iv_crequest_id = lv_crequest_id

                                 it_key         = lt_entity ).


  CATCH cx_usmd_gov_api_core_error cx_usmd_gov_api.
    "Possibility to handle the erroneous data or go on.
ENDTRY.

"10th: Save the change request (and the entity data of course)
TRY.
    lo_gov_api->save( ).
    "Save is done in draft mode by default so it is possible to

    "save the change request even if change request data or

    "entity data is not consistent.
  CATCH cx_usmd_gov_api_core_error.
    EXIT.
    "Adequate Exception handling
ENDTRY.

"11th: At the end, it is necessary to clean the house
TRY.
    lo_gov_api->dequeue_entity(   EXPORTING iv_crequest_id = lv_crequest_id
                                            iv_entity_name = 'CARR'
                                            it_data        = <lt_carr_key> ).
    lo_gov_api->dequeue_crequest(

                        EXPORTING iv_crequest_id = lv_crequest_id ).
  CATCH cx_usmd_gov_api.
    "Not a tragedy - maybe the workflow could not be processed properly after
    "it was started
```

```
ENDTRY.

COMMIT WORK AND WAIT.

"12th: If everything is fine, the workflow can be started for
"the change request (this is like a 'submit')
TRY.
    lo_gov_api->start_workflow( iv_crequest_id = lv_crequest_id ).
  CATCH cx_usmd_gov_api_core_error.
    "Adequate Exception handling
ENDTRY.

"Interested in the errors occured?
lt_messages = lo_gov_api->get_messages( ).
```

5.3 Code Example – Change Flight Connection (with governance API)

```
*&---------------------------------------------------------------------*
*& Report  ZCHANGEPFLI_SF
*&---------------------------------------------------------------------*
*& This example creates a new change request. Along with the change
*& request, an existing flight connection is changed. Additionally, a
*& dependent entity of type flight schedule is changed/created.
*&
*& The exceptions raised by the governance API contain the error
*& messages and some other attributes. Therefore, it is possible to react
*& to the errors raised by the governance API.
*&---------------------------------------------------------------------*

REPORT zchangepfli_sf.


DATA:
lo_gov_api           TYPE REF TO if_usmd_gov_api,
lv_crequest_id       TYPE usmd_crequest, "Change Request ID

lr_pfli_key_str      TYPE REF TO data,  "Entity Flight Connection - key structure
lr_pfli_key_tab      TYPE REF TO data,  "Entity Flight Connection - key table
lr_pfli_data_str     TYPE REF TO data,  "Entity Flight Connection - data structure
lr_pfli_data_tab     TYPE REF TO data,  "Entity Flight Connection - data table

lr_flight_key_str    TYPE REF TO data,  "Entity Flight - key structure
lr_flight_key_tab    TYPE REF TO data,  "Entity Flight - key table
lr_flight_data_str   TYPE REF TO data,  "Entity Flight - data structure
lr_flight_data_tab   TYPE REF TO data,  "Entity Flight - data table

ls_entity            TYPE usmd_gov_api_s_ent_tabl,
lt_entity            TYPE usmd_gov_api_ts_ent_tabl,

lt_messages          TYPE usmd_t_message.

FIELD-SYMBOLS:
<ls_pfli_key>        TYPE any,
<lt_pfli_key>        TYPE INDEX TABLE,
<ls_pfli_data>       TYPE any,
<lt_pfli_data>       TYPE INDEX TABLE,

<ls_flight_key>      TYPE any,
<lt_flight_key>      TYPE INDEX TABLE,
<ls_flight_data>     TYPE any,
<lt_flight_data>     TYPE INDEX TABLE,

<value>              TYPE any.

"1: Create an instance of the governance API
TRY.
    lo_gov_api = cl_usmd_gov_api=>get_instance( iv_model_name = 'SF' ).
  CATCH cx_usmd_gov_api.
    EXIT.
ENDTRY.

"2: Create the data references needed to maintain the flight connection entity
"Create a data reference of the key structure/table of entity PFLI
lo_gov_api->create_data_reference(
      EXPORTING iv_entity_name = 'PFLI'
                iv_struct      = lo_gov_api->gc_struct_key
      IMPORTING er_structure   = lr_pfli_key_str
                er_table       = lr_pfli_key_tab ).

lo_gov_api->create_data_reference(
      EXPORTING iv_entity_name = 'PFLI'
                iv_struct      = lo_gov_api->gc_struct_key_attr
      IMPORTING er_structure   = lr_pfli_data_str
                er_table       = lr_pfli_data_tab ).

"Assign the created data references for the flight connection key to the field symbols
ASSIGN lr_pfli_key_str->* TO <ls_pfli_key>.
ASSIGN lr_pfli_key_tab->* TO <lt_pfli_key>.

ASSIGN COMPONENT 'CARR' OF STRUCTURE <ls_pfli_key> TO <value>.
<value> = 'LH'.
ASSIGN COMPONENT 'PFLI' OF STRUCTURE <ls_pfli_key> TO <value>.
<value> = '0401'.
INSERT <ls_pfli_key> INTO TABLE <lt_pfli_key>.

"Assign the created data references for the flight connection data to the field symbols
ASSIGN lr_pfli_data_str->* TO <ls_pfli_data>.
ASSIGN lr_pfli_data_tab->* TO <lt_pfli_data>.

"3: Create a new change request using change request type and a
description (required)
TRY.
    lv_crequest_id = lo_gov_api->create_crequest(
                     iv_crequest_type = 'SFP02'
                     iv_description   = 'Change Flight Connection LH 400' ).

  CATCH cx_usmd_gov_api.
    "Something went wrong while creating the change request (e.g. data model blocked
or change request type unknown).
    EXIT.
ENDTRY.

"4: Before making changes to an object, the object needs to be enqueued.
TRY.
    lo_gov_api->enqueue_entity( EXPORTING iv_crequest_id = lv_crequest_id
                                          iv_entity_name = 'PFLI'
                                          it_data        = <lt_pfli_key> ).
  CATCH cx_usmd_gov_api_entity_lock cx_usmd_gov_api.
    EXIT.
ENDTRY.

"5: Read the flight connection data in order to do some changes
TRY.
    lo_gov_api->read_entity( EXPORTING iv_crequest_id   =     lv_crequest_id
                                       iv_entity_name   =     'PFLI'
                                       it_key           =     <lt_pfli_key>
                             IMPORTING et_data          =     <lt_pfli_data> ).
  CATCH cx_usmd_gov_api_core_error cx_usmd_gov_api.
    EXIT.
ENDTRY.

READ TABLE <lt_pfli_data> INDEX 1 INTO <ls_pfli_data>.
CLEAR <lt_pfli_data>.

ASSIGN COMPONENT 'ARRTIME' OF STRUCTURE <ls_pfli_data> TO <value>.
<value> = '075500'.
ASSIGN COMPONENT 'DEPTIME' OF STRUCTURE <ls_pfli_data> TO <value>.
<value> = '184000'.
INSERT <ls_pfli_data> INTO TABLE <lt_pfli_data>.

"6: Write the changes for the flight connection
TRY.
    lo_gov_api->write_entity( EXPORTING iv_crequest_id = lv_crequest_id
                                        iv_entity_name = 'PFLI'
                                        it_data        = <lt_pfli_data> ).
  CATCH cx_usmd_gov_api_entity_write.
    EXIT. "Do better next time!
```

```
ENDTRY.

"7: Create all the data references needed to maintain the flight entity
"Create a data reference of the key structure/table of entity FLIGHT (Flight)
lo_gov_api->create_data_reference(
      EXPORTING iv_entity_name = 'FLIGHT'
                iv_struct      = lo_gov_api->gc_struct_key
      IMPORTING er_structure   = lr_flight_key_str
                er_table       = lr_flight_key_tab ).

lo_gov_api->create_data_reference(
      EXPORTING iv_entity_name = 'FLIGHT'
                iv_struct      = lo_gov_api->gc_struct_key_attr
      IMPORTING er_structure   = lr_flight_data_str
                er_table       = lr_flight_data_tab ).

"Assign the created data references for flight connection key to the field symbols
ASSIGN lr_flight_key_str->* TO <ls_flight_key>.
ASSIGN lr_flight_key_tab->* TO <lt_flight_key>.

MOVE-CORRESPONDING <ls_pfli_key> TO <ls_flight_key>.
INSERT <ls_flight_key> INTO TABLE <lt_flight_key>.

ASSIGN lr_flight_data_str->* TO <ls_flight_data>.
ASSIGN lr_flight_data_tab->* TO <lt_flight_data>.

"8: Read some flight data in order to do some changes
TRY.
    lo_gov_api->read_entity( EXPORTING iv_crequest_id   =     lv_crequest_id
                                       iv_entity_name   =     'FLIGHT'
                                       it_key           =     <lt_flight_key>
                             IMPORTING et_data          =     <lt_flight_data> ).
  CATCH cx_usmd_gov_api_core_error cx_usmd_gov_api.
    EXIT.
ENDTRY.

READ TABLE <lt_flight_data> INDEX 1 INTO <ls_flight_data>.
CLEAR <lt_flight_data>.

IF sy-subrc <> 0.
  MOVE-CORRESPONDING <ls_flight_key> TO <ls_flight_data>.
  ASSIGN COMPONENT 'FLDATE' OF STRUCTURE <ls_flight_data> TO <value>.
  <value> = '31122013'.
ENDIF.

ASSIGN COMPONENT 'SEATSOCC' OF STRUCTURE <ls_flight_data> TO <value>.
<value> = '209'.

INSERT <ls_flight_data> INTO TABLE <lt_flight_data>.

"9: Write the changes for the flight
TRY.
    lo_gov_api->write_entity( EXPORTING iv_crequest_id = lv_crequest_id
                                        iv_entity_name = 'FLIGHT'
                                        it_data        = <lt_flight_data> ).
  CATCH cx_usmd_gov_api_entity_write.
    EXIT. "Do better next time!
ENDTRY.

"10: The complete change request should be checked before it is saved
TRY.
    lo_gov_api->check_crequest_data( iv_crequest_id = lv_crequest_id ).
    "Collect the entities to be checked
    ls_entity-entity = 'PFLI'.
    ls_entity-tabl   = lr_pfli_key_tab.
    INSERT ls_entity INTO TABLE lt_entity.
    "Check the entity
    lo_gov_api->check_complete_data(
                           EXPORTING iv_crequest_id = lv_crequest_id
                                     it_key         = lt_entity ).

  CATCH cx_usmd_gov_api_core_error cx_usmd_gov_api.
    "Handle the erroneous data or go on.
ENDTRY.

"11: Save the change request (and the entity data, of course)
TRY.
    lo_gov_api->save( ).
    "Save is done in draft mode by default so it is possible to
    save the change request even if the change request data or
     the entity data is not consistent.
  CATCH cx_usmd_gov_api_core_error.
    EXIT.
    "Adequate exception handling
ENDTRY.

"12: At the end, it is necessary to clean the house
TRY.
    lo_gov_api->dequeue_entity(   EXPORTING iv_crequest_id = lv_crequest_id
                                            iv_entity_name = 'PFLI'
                                            it_data        = <lt_pfli_key> ).
    lo_gov_api->dequeue_crequest( EXPORTING iv_crequest_id = lv_crequest_id ).
  CATCH cx_usmd_gov_api.
    "Adequate exception handling
ENDTRY.

COMMIT WORK AND WAIT.

"13: If everything is correct, the workflow can be started for
the change request (this is like a 'submit')
TRY.
    lo_gov_api->start_workflow( iv_crequest_id = lv_crequest_id ).
  CATCH cx_usmd_gov_api_core_error.
    "Adequate exception handling
ENDTRY.

"Interested in the messages occurred?
lt_messages = lo_gov_api->get_messages( ).
```

6 Copyright