PUBLIC

SuccessFactors Integrations: Best Practices using SAP SuccessFactors APIs for Custom Integrations

pwc

Aspire HR
Right People, Right Results

THE BEST RUN SAP

Document Details

Name	Objective	Audience
SuccessFactors Integrations: Best Practices using SAP SuccessFactors APIs for Custom Integrations	Give guidance to partners to build better custom integrations using SuccessFactors APIs and help customer and partners to review and spot bad patterns in legacy integrations.	SAP SuccessFactors Customers: IT and HR professionals; SAP SuccessFactors Implementation Partners: Consultants, solution architects and project managers

Change Log

Version	Date	Description
1.0	01.02.2020	Initial version
1.1	21.02.2020	Document was changed to incorporate following minor/major changes:
1.2	13.05.2020	Template adjustment and reference updated
1.2.1	18.01.2021	Fixed broken links and cross references, formatting
1.2.2	23.03.2021	Name changes applied (CPI,SCP to SIS and BTP)

Supported Releases

Product	Release - From	Release-Valid till
SAP SuccessFactors Employee Central	1911	

Contribution

Role	Name	Organization
Author / Owner	SAP SuccessFactors Product Management	SAP SE
Contributer	Andreea Bejenaru	PwC
Contributor	Belinda Lineman	AspireHR
Contributor	Clarisse Gondoprawiro	PwC
Contributor	Marc van Driel	Comentec

The recommendations in this document are based on the functionality available up to SAP SuccessFactors release mentioned above. Future functionality can impact the recommendations provided by this document. We strive to keep these recommendations up-to-date, however, in case you find that recent new functionality has not yet been considered in the latest version of this document, please reach out to your Customer Success Manager / Partner Delivery Manager or send an email to SAPSuccessFactorsIDPDoc@sap.com.

Implementation Design Principles (IDPs) for SuccessFactors solutions are delivered by SAP for helping customers and partners on how to choose the most appropriate strategy and solution architecture for SAP SuccessFactors implementations. IDPs are compiled taking into consideration the experience of many implementation projects and addressing frequent business requirements as well as real-life implementation challenges. They are continuously reviewed and updated as product functionality evolves. In addition, the reader is advised to read and familiarize with essential and additional product-related documentation which includes Implementation Guides, SAP Notes, SAP Knowledge Base Articles, and additional assets as referenced in this document. see chapter 8.

TABLE OF CONTENTS

1. TERMINOLOGY

The following table explains some abbreviations used in this document.

Abbreviation	Description
EC	Employee Central
ERP	SAP Enterprise Resource Planning often referred in the document pertains to SAP HCM on premise system
MDF	Meta Data Framework
RBP	Role Based Permissions
UI	User Interface
A2A, B2B, B2G	According to (Allgaier, 2019) there are four **Integration Styles**, Process Integration, Data Integration, User Integration and Thing Integration. Within those Integration Styles are different **integration use case patterns** such as A2A, B2B and B2G for process integration or master data synchronization or data virtualization for data integration.
SOAP	Simple Object Access Protocol (SOAP) is a **messaging protocol** specification for exchanging structured information in the implementation of web services in computer networks. (SOAP, 2019)
REST	Representational state transfer (REST) is a **messaging protocol** that defines a set of constraints to be used for creating Web services. Web services that conform to the REST architectural style, called RESTful Web services, provide interoperability between computer systems on the Internet. RESTful Web services allow the requesting systems to access and manipulate textual representations of Web resources by using a uniform and predefined set of stateless operations. (Representational state transfer, 2019)
OData	OData (Open Data Protocol) is an ISO/IEC approved, OASIS standard for **a messaging protocol** that defines a set of best practices for building and consuming RESTful APIs. OData helps you focus on your business logic while building RESTful APIs without having to worry about the various approaches to define request and response headers, status codes, HTTP methods, URL conventions, media types, payload formats, query options, etc. (OData.org, 2018)
	Messaging modes or communication modes can be either synchronous or asynchronous. In a synchronous communication the initiator of request message (client) waits for the receiver of the message (server) to return a response message.
	Messaging direction or communication directions from a system point of view can be either inbound or outbound. Outbound communications are also known as Events or Push Messages. In some documents inbound and outbound are also used to define the data flow direction, e.g. if data is getting out of the system (query, events) or into the system (write). In this document we use inbound and outbound only in the context of communication, i.e. if a system calls (outbound) or is being called (inbound).
	The **extraction pattern** defines the scope of data being extracted from a system to keep two copies of the transferred information in sync. Extraction patterns can be either a **full load** or a **delta load**. In a case of a delta load we distinguish between **record level** and **field level delta loads**. Objects are composed of atomic entities or resources. An employee object for example is composed out of atomic entities such as Emails, Compensation Information which have the same data structure.

Abbreviation	Description
Object, Entity, Record, Field	Within this document we refer to an **entity** as a metadata definition of for collection of records with identical structured data with a common business key. This data is usually stored in a single data base and exposed by an API. Each given **business key** identifies a single **record** of the same data structure within the database of the entity. Each entity consists of **properties** which can have different **data types**. Same fields of different records have the same data type. Multiple entities can be combined or joined into an (business) object, Email Information and Employment Information for example are entities which are part of the object Employee. In case of multiple entities forming a hierarchy we call the top most one the **root entity**.
Deep Filter	In OData we refer to a deep filter in case the root entity is not filtered but any of the sub entities. In such a case the filter condition includes the name of a field and the corresponding navigation path, e.g. $filter=navSubPath1/navSubPath2/FieldName+EQ+'Value'.
Proprietary URL Parameter	OData has predefined URL parameters for filtering, selecting or paging, e.g. $filter,$select, $top. All of those standard URL parameter start with a $. If parameters are required which are not part of the standard we do not use a $ and call them proprietary URL parameters, e.g. recordStatus, fromDate, asOfDate.
Function Import	A function import is a specific URL which does not support CRUD operation or standard oData parameters such as $filter or $select. It is a custom function with non-standard URL parameters to perform a specific task, this can be reading special information such as permission data or performing an action such as approving an workflow.
API	Application Programming Interface is a communication protocol used to enable communication between systems. Typical types of those communication protocols are SOAP, REST or OData.
CE API	Compound Employee API, a SOAP based API reading Employee Information from Employee Central supporting different extraction patterns such as full loads and delta loads.
Pagination	There are different paging options possible to avoid that API calls result in timeouts. Those are **client sided paging** and **server sided paging**. In the case of client sided paging the client (initiator of the communication) tells the server which page and how many records of the page to return, while in case of server sides paging this is done by the server.
TCO	Total cost of operation (TCO) defines the cost of a vendor or customer to operate his product or custom integration.
TCD	Total cost of development (TCD) defines the cost to develop a product, e.g. a custom integration.
MDF	Metadata framework (MDF) is a framework in SuccessFactors to create own code free custom entities including persistency, UI and APIs. MDF is also used internally by SuccessFactors. In this document we sometimes distinguish between entities and APIs based on MDF and EmployeeCentral to highlight additional features which are available for the corresponding class of entities.
Audit	When we talk in this document about audits we refer to audit records being created to track all changes and deletions done to a given entity or set of entities. Those audit log (sometimes also referred to as change logs or change pointers) can be used to track all changes done and allow us to build delta replications where data is only transferred in case there have been changes. Different from the audit logs, there are also API logs which keep track of all called APIs.
Hard and Soft Delete	There are always two options to define a deleted object, use a status field indicating the deletion or really delete the record from the data base. In this document we refer to the first approach as a soft delete while the 2nd one is defined as a hard delete.

2. ABSTRACT

This document provides advice on the technical design of SuccessFactors custom integrations using OData APIs and Compound Employee APIs. As such it covers guidelines for the following topics:

- Picking the right API for different use cases
- Delta and deletion handling
- Handling date/time data with API
- Batching API request
- Accessing workflow data
- Writing data into Employee Central

3. INTRODUCTION

Understanding APIs and how they ought to be used is the basis of all successful integrations.

The following list summarizes what will be covered by this document:

- Avoid deprecated APIs or APIs which have not been intended for a specific use-case.
- Make use of API and/or middleware features to keep the number of API calls and the amount of transferred data low, e.g. delta loads, paging, batching or field level selects.
- Make use of reliable integration patterns with respect to pagination and error handling.
- Use the APIs in the right way to read effective dated data and deletions.
- Write data into SuccessFactors in a consistent, transactional and performant way.

Following the guidelines provided by this document will help you to avoid shortcomings in the integration processes spanning from performance and stability issues to data inconsistencies.

This IDP mainly focuses on process and data integrations styles for Employee Central rather than IoT or user integration styles such as Desktop, Mobile or UI Integration.

This document is the first document in a series of documents addressing custom integrations. While this document addresses mainly the API usage from middleware platforms in general, the other IDPs will focus on building and running custom content on SAP Integration Suite in specific. Following the recommendations in those documents shall result in better integrations with respect to most integrations qualities such as performance, robustness, stability and consistency.

4. BUSINESS REQUIREMENT

4.1. Functional Requirements

This chapter defines the functional or business requirements for the different areas we discovered as pain points in the initial IDP workshops with more than 20 partners and experts, while the next chapter covers non-functional or technical requirements such as performance or security.

4.1.1 Using the right SuccessFactors API

When it comes to use the right SuccessFactors API (SF API) one questions should always be: What is the Integration Style? (Allgaier, 2019) distinguishes four integration styles: process, data, user and IoT integrations. Those can be mapped to integration technologies such as our SF APIs. In this document we focus mainly on process and data integration styles and explain which API shall be used and how.

We will give a high-level guidance on all integrations styles and which API to use but also do a detailed recommendation for process and data integration styles to enable partners to build good integrations.

4.1.1.1 Master Data Synchronization

One integration use case pattern of the data integration style is the Master Data synchronization. Many partners build custom integration to transfer SuccessFactors data to 3rd Party Application. Since those two systems have been designed independently from each other, they are usually not compatible with respect to their data structures or their API behavior. As a result, there must be an active part in the middle (or in the 3rd party application) which maps those differences.

In order to limit the effort for this mapping, partners and customers ask of course for different available protocols and extraction patterns from the SuccessFactors APIs which are as close as possible to their 3rd system they want to integrate. They ask for this to keep the effort for the creation of the integration low (TCD) but also the performance high and the transferred data low (TCO).

Common required data extraction patterns beside an obvious full load are (business) object level delta, record level delta and field level delta for single as well as for joined entities. Those kind of delta modes will be used to minimize the amount of transferred data by transferring only changes instead of all data but still allow to be close to what the target system expects, e.g. all records from all entities of an employee in case there is any change (object delta for employee) but not all employees (full load). We will give guidance in the detailed solution chapter with respect to the three different extraction patterns and which API to use based on available features and a good mix of TCO and TCD.

4.1.1.2 Keep the result set small and avoid unnecessary calculations for good performance

Beside delta handling there are more ways to improve the performance of integrations by providing (and using) corresponding features in the APIs. Those include:

- Reuse Sessions in subsequent APIs calls instead of repeating the authentication all the time
- Allow bypassing of end-user permission checks in case of master data synchronizations for technical users
- Suppress triggering of rules, if not required
- Offer multiple filter options and parameters in the APIs to retrieve only required data, but also avoid too complicated and too many deep filters too
- Offer the capability to select the fields to be retrieved and the joins to be done
- Offer the capability to join the data in the backend of the source system
- Batching multiple independent API calls into one
- Efficient sever based paging

All those features have in common that they help to reduce the computation time on the backend or the middleware and reduce the amount of transferred data. Both KPIs (time and size) are often limited in the different systems but also in the infrastructure between those. Hence knowing and using those API features to keep within those restrictions is crucial to build robust and reliable integrations

4.1.1.3 Event driven integrations

While data extraction patterns as those above are realized by an inbound communication pattern which are usually scheduled there is also the need in some projects to get changed information pushed into another system and do not wait till the next replication is scheduled. Use cases from customer projects are for example document generation in case of data changes or immediate creation or deletion of employees in 3rd party system in case of hires or terminations. We will explain shortly how and when to use those and explain the challenges and best practices for those.

4.1.1.4 Timeouts, Size and other Limits

When using http or https as a synchronous communication protocol there is always a maximum time when a long running session is going to run into a timeout. This time is not fixed. It is defined by the minimum time of all involved software and hardware components from the sender system to the receiving system. In addition to this time-based limitation there is also a size-based limitation for the length of the URL and the payload of the request and the response. As a result, it is required to offer and use reliable paging mechanism to keep the payload and the time for the APIs calls small. We are listing many of those restrictions also in the Appendix in chapter 8.1 and explain how to avoid them.

4.1.2 Batching SuccessFactors OData calls

When building integrations using OData or REST as a communication protocol there are certain URL length limitations which might cause an API call to fail. In such a case the only way to continue using the API is to move the API call from the URL into the payload using $batch. This might happen especially in cases where in data replication scenarios many entities and many fields are requested and the corresponding $filter, $select and $expand URL parts are long.

While it is a valid requirement to simplify multiple API calls in a simple single API call, e.g. create a hire new employee API by putting all entities in $batch (see also chapter 5.2.3), it is important to know that those capabilities to define a new simplified API using $batch are limited.

4.1.3 Writing data into Employee Central (EC)

Employee Central offers several capabilities to write data. From a technology point of view these include OData, file-based CSV and SOAP. This chapter explains different write use cases and their requirements from a functional point of view.

There are several use cases when writing into EC is required

- Data migration during initial setup of the system
- Master data replications from a 3rd party HR system which acts as a master while Employee Central is a slave. Partial/mixed scenarios included, where some part of the employee base is managed in EC and another one on a 3rd party system.
- Process Integration into 3rd party systems, e.g. creating a ticket in a ticketing system to ensure that equipment is ordered and installed for a new employee

Those three use cases differ from their requirements towards the APIs. For the data migration the most important aspect is to get a lot of employees into the system in a very short period of time. Inconsistencies of created employees are fine as long as they can be tracked and fixed as part of the migration process, since the system is not productive, yet this does not have any consequences.
The master data replication is scheduled on a regular basis usually in production on a daily or hourly basis. As a result, it is important to avoid inconsistencies. If the amount of data replicated is small consistency might be more important than performance, especially if there might not be a proper monitor available to track those errors in case of custom integrations.[1]
In case of process integrations similar arguments are true, again consistency and transactional behavior is more important than throughput and triggering of business rules might become more important.

Use Case	Transactional Behavior and Consistency	Throughput	Business Rules	Permissions	Workflow	Update Changes Only	Purge
Data Migration	Not important	High throughput required	Only in exceptions	No checks required	No	No	Yes

[1] For SAP SuccessFactors delivered productized integrations such as the SAP ERP HCM to SAP SuccessFactors Employee Central there is an detailed error monitor available, it was built as part of the those integrations.

Master Data Replication	Important requirement	Less important	Not if data comes already with right values	No checks required	No, usually done in the master	Yes	Should be avoided
Process Integration	Important requirement	Less important	Yes, also depending on the detailed use case	Yes, depending on the detailed use case.	Yes, depending on the use case	Only if amount changed records big	Should be avoided
UI Extension	Yes, very important	Not that important	Yes	Yes	Yes	No, proper change tracking in UI	No

Triggering of business rules is less a requirement for the migration and master data replication use case since in both scenarios the 3rd party source system should take care about the correct calculated values instead of relying on rule execution in the target system, if possible. This is usually different for process integrations and UI extensions. Both use cases might require that additional business logic defined in rules is triggered in the Employee Central system.

User based permission handling is usually only required for write APIs in case of UI Integration (Extension Scenarios) or real time process integration where instead of technical users the named user of the logged-on Person is used.

Having the capability to change only those records in Employee Central which need a change is an important feature to avoid an avalanche effect. In case the source system cannot send changes only but sends more data than there have been changes into Employee Central the result will be that also more data is replicated from Employee Central to other systems connected to it. To avoid this effect, a comparison of the payload of the write API with the payload in Employee Central shall help to reduce this effect if required. This is usually only needed if many records are updated, e.g. in a master data replication.

Triggering of approval workflow is another feature which is usually not required in a data migration or master data replication case. In case of data migration, data does not have to be approved while in case of the master data replication it was most probably already approved in the source system using a workflow there. Only for process integration use cases and UI extensions this might make sense.

Data migration is often an iterative process which requires data cleansing, repeated migration of data and sometimes even a complete refresh. In those cases where the file was dirty and there have been a lot of errors it is often better not to try to clean up but just delete again all records and start from scratch with a new full load. In that case it is important to have a feature which allows to purge all records before creating new ones to avoid that the result is a mixture of new and old records. For similar reasons it might come handy in other cases as well to avoid complicated handling of existing versus new and to be deleted records.

In chapter 5.3 we will explain the different API features SAP SuccessFactors offers for the three use cases above and best practices how to use them.

4.1.4 Managing Dates, Times, Time zones and Daylight saving

There are multiple dimensions of times and dates to be handled in the HR Data and hence must be retrieved and written by APIs in a correct way. Those are effective dates and times, change dates and times and business dates and times.

All data in a HR system must have information about the date, time and time zone when a record was created and last changed. The time zone information can be explicit or implicit, time zone is implicit if store by default in UTC or a fixed and know server timezone. For technical but also for audit reasons, it is important to keep

track of those changes, including deletions and multiple changes of the same record[2]. This data is being maintained in audit tables. Those tables with deletion and delta information about all changes are not visible to the normal business user, he/she can only see the current version of the data but not all interim versions.

For all data which can change over time and where it is important from a business point of view to keep and know this history, we require to data to be stored using effective start and end dates. Those dates are also called floating dates and do not need time or timezone information, doing conversion here would result in unwanted date shifts. In case multiple changes for a single day are possible a further sequence number is required. Examples of such is a Job History or the Compensation History. Other data, such as Email or Phone Information is not required to store such history. As a result, those effective dates and sequences are often also business keys of the OData Entity. Those effective dated records can also have audit tables as mentioned before. Each effective dated record might have been created, updated, deleted and created again for the same business key (user Id, start date and sequence number) multiple times, the end user just sees the last version of this record while all the changes are stored in the audit tables.

In addition to those two major dimensions effective date of a record and the change time of a record there are sometimes additional time fields within a record to define business relevant date and time information for example the birthdate. We refer to those as business dates and times.

If a timestamp is included, it is important to know for all those three types of fields, in which time zone this time is stored, to enable comparison and ensure consistency. Especially if data is accessed with APIs and stored in other places or written from another place. Without knowing the right time zone conversion errors would be the consequence. It is also important to know if a field has a time information or is only a pure date field to avoid conversions for those.

With respect to the change date and time this becomes important to ensure that in case of delta replication, where only changed data is being read and synchronized to a target system, no records are being lost or transferred several times.

We will explain the difficulties with that time information in chapter 5.4 and explain how to avoid common pitfalls and create correct integrations with respect to time and date handling.

4.1.5 Performing and Detecting deletions

Performing deletions in an HR System is risky for several reasons. First, deletions can cause data inconsistencies if records of an entity “A” are still referring to the deleted records of entity “B”. In addition, those hard deletes cause issues in data integrations, especially for master data synchronization. In order to ensure that deleted records are deleted in other systems as well we must retrieve the deleted objects and perform deletions in all relevant systems. Again, if in this system the record is being used a deletion might not be possible or cause data inconsistencies again. For that reason, soft deletes, setting the status of an object to deleted or inactive but keeping it in the DB, should always be the preferred solution.

In some cases, a hard delete is wanted or cannot be avoided. We talk about hard deletes here in the context of a business object rather than a row in the database, e.g. effective dated record of a business objects will usually always be hard deleted while the whole business objects, e.g. cost center can be either soft deleted (set to inactive) or all effective dated records can be deleted (hard delete). The challenge being addressed in this section of the document is how to perform such kind of hard deletions and how to make use of existing features in our APIs to detect those and build reliable integrations which also keep deleted records between systems in sync.

[2] Creation of audit records shall be enabled only if there is a clear business need for this, e.g. in case of integrations needing this data in APIs. Having audit logs being created without a need increases the data volume in the system and slows it down.

4.1.6 Accessing pending Data in EC Workflows with APIs

Below are two use cases from a multiple customer project being run by one of our partners. They all have in common the need to access and process workflow data:

Requirements for use case HR reporting:

- From the perspective of HR reporting, customers really want to report on pending workflows i.e. how many workflows are in the backlog, what are users mainly requesting for via workflows. With knowing this, clients could improve their efficiency of timely handling of workflows.
- Position Approval Workflow, so that they can show e.g. their board an overview of all pending requests on new positions, before they are actually created in the system
- Even though there is a possibility to report on pending workflows in report center, in some cases customers are looking for custom solutions based on APIs

Requirements for third-party integration use cases:

- A ticket management system such as ServiceNow, or others: If the ticketing system could access this pending workflow data, then workflows could be attached to official tickets and really enable the HR service desk of clients. In addition in some cases it might make sense also to approve the workflow, or a workflow step from the ticket management system.
- Integration into an E-File system: One use case is for employee data changes which require attachments and are associated with workflows such as personal info changes which use documents such as marriage certificates, proof of address changes, etc. With this pending workflow API, the e-file system could possibly directly access these attachments associated to the pending workflow.
- Document generation: Create a contract amendment for example (e.g. change of working time, change of location) with the data that are in the pending workflow and not wait until the end of the workflow. Documents can be created before the last approval step and before the data are visible in the system.

4.2. Technical Requirements

This chapter explains common technical or non-functional requirements for pain points being addressed. Those include security, performance, documentation and support.

4.2.1 Using the right SuccessFactors API

4.2.1.1 Security

In a cloud environment the most important nonfunctional requirement is security. Systems and their APIs must provide corresponding security mechanism such as IP-Filtering, authentication (e.g. oAuth in addition to basic auth) but also proper permission handling. While those are important to avoid misuse, having the capability to monitor API usage can add an additional level of security but also help to analyze incorrect usage of APIs which result in bad performance.

4.2.1.2 API Documentation and Discovery

Another important non-functional requirement for APIs is good documentation and easy discovery. Missing proper knowledge about APIs results in wrong usage with negative consequences for the customer project and operations of the vendor while not knowing that the API exists might harm our ecosystem and the reputation of our product as an open and flexible platform.

4.2.1.3 Communities, Support and Feature request

Even with the best documentation there are cases where documentation cannot cover every use case, hence it is also important to have an active community and a support channel to address those needs.

Finally, missing APIs or API features need another channel to get addressed.

4.2.2 Batching SuccessFactors OData calls

From a non-functional point of view the major use case for batching OData request is avoiding roundtrips and hence also execution time to gain better performance. This is especially true for user interface integrations but also for process integrations.

Batching allows to combine several requests in one API when a navigation using OData $expand is not possible. There are several cases where a combination of multiple API calls is wanted:

- The API to be used does not offer a filter condition with multiple values but with single values only but it is the requirement to process this API for multiple filter values, e.g. approving EC Workflow request or reading pending data
- The APIs to be used are not linked to each other or do not offer a navigations path, e.g. reading data for Departments of a given country and in parallel also reading cost centers with a specific number range

4.2.3 Writing data into Employee Central (EC)

Depending on the use cases listed in chapter 3.1.4 there are differences in the non-functional requirements with respect to performance (both response time and throughput) and security. Beside those differences, there are no further non-functional requirements addressed in this document.

4.2.4 Managing Dates, Times, Time zones and Daylight saving

There are no specific non-functional requirements to be considered.

4.2.5 Performing and Detecting deletions

There are no specific non-functional requirements to be considered.

4.2.6 Accessing pending Data in EC Workflows with APIs

There are no specific non-functional requirements to be consider for handling EC workflows with APIs.

5. SOLUTION OVERVIEW & CONCEPTS

In this chapter we are giving a high-level overview what is available as of today in the SAP SuccessFactors HXM Suite and how this technology should be used before explaining the details in chapter 5.

5.1 Using the right SF API

SuccessFactors is offering two major API Technologies. SOAP based SFAPI is a legacy API and deprecated as of August 1, 2018 with the exception of Compound Employee API. In all other cases the OData API shall be used.

Overview ODATA vs. SOAP

The SFAPI is SuccessFactors Data API. It is a SOAP-based Web Service designed for importing and exporting data to and from your SuccessFactors instance. It provides generic CRUD (Create, Read, Update, Delete) operations to access data, as well as meta-data operations to allow runtime discovery of the data. Data is

exposed as entities called SFObjects, which are conceptually analogous to database tables. Using the meta-data operations, you can list the SFObjects available to the API, and describe the fields in these entities. Using the CRUD operations, you can query or edit the data. (SAP, 2020)

For more information on SFAPI, you can consult the SFAPI Developer guide:
https://help.sap.com/doc/0bd2f6fea0154ac6aacd44f1cacbfc71/latest/en-US/SF_HCM_SFAPI_DEV.pdf

The Open Data Protocol (OData) is a standardized protocol for creating and consuming data APIs. OData builds on core protocols like HTTP, and commonly accepted methodologies like REST. The result is a uniform way to expose full-featured data APIs. OData provides both a standard for how to represent your data and a metadata method to describe the structure of your data, and the operations available in your API. SuccessFactors OData API service is based on OData V2.0. The HCM Suite OData API is SuccessFactors Web Services API based on OData protocol intended to enable access to data in the SuccessFactors system. The API is data oriented. This API provides methods for CRUD style access (Create, Read, Update and Delete). The API is best used for frequent or real time requests for small amounts of data. Large data requests are better handled by batch FTP processes. Each SuccessFactors module can be accessed using its own set of entities. (SAP Help Portal, 2019)

For more information on Employee Central Odata APIs, you can consult the OData Reference Guide for EC: https://help.sap.com/doc/7efdca36492e47c7b20ab92c4ca6323c/latest/en-US/SF_EC_OData_API_REF.pdf

The simple SFAPIs and AdHoc APIs are deprecated as of 1 August 2018 – but this does not include the Compound Employee API. For more information about deprecation:
https://help.sap.com/viewer/1a981b4253d849eb82d4ca32bc767750

From an integration style point of view we can clearly state that the OData APIs are the only ones suitable for user-based integrations while the SOAP based Compound Employee API was especially designed to cover the data integration style and in specific the master data synchronization of employee master data. As a result, the fallback in all other cases is again OData. The remaining integration technology of CSV file based scheduled import and export to SFTP shall be used in those cases only where SFTP cannot be avoided, e.g. if the receiving system, or sending system in case of import, must make use of SFTP and does not provide any APIs.

With respect to data being extracted the two APIs have a major difference. While OData covers almost all Employee Central Data including write and read in addition to entities such as Picklist, Users, Workflows, the Compound Employee API is limited to read employee master data (person and employment entities). Both support custom objects defined by our MDF generic object model, but the Compound Employee API can cover only employee based MDF objects having the User as external code.

5.2 Batching SuccessFactors OData calls

$batch is a simple way offered by the OData Standard to move multiple API calls into the payload/body of a single $batch API call.

The OData V2 Specification describes the structure of $batch in detail:
https://www.odata.org/documentation/odata-version-2-0/batch-processing/

While our SAP SuccessFactors documentation gives examples of some calls:
https://help.sap.com/viewer/d599f15995d348a1b45ba5603e2aba9b/latest/en-US/869a37d42a2849d6bc8cda597df18bd1.html?q=$batch

$batch, if implemented by the corresponding API server, only works within the same domain. Cross domain $batch is not supported. This has to be considered before combining multiple API calls into one $batch call.

5.3 Writing data into Employee Central (EC)

Employee Central offers high level two options to write data, file or API based. The file-based approach of uploading data into EC can be done manually from a UI or scheduled via Provisioning or Integration Center to read data from a SFTP sever and import it, see details here:

Import Documentation: https://help.sap.com/viewer/f5c753ba58814ef0ab181747824c41ed/latest/en-US

Integration Center: https://help.sap.com/viewer/60ba370328e0485797adde67aee846a0/latest/en-US

There are also API based integrations available. Since SOAP as a technology is deprecated the only viable options to write into EC using APIs is OData. In OData there are again two different options available. Either the OData core operations to create and updated records using PUT and POST (see OData V2 Spec here: https://www.odata.org/documentation/odata-version-2-0/operations/) or an additional comfort feature called UPSERT. This feature is implemented in OData as a function import and is called as a POST request using this pattern:

POST odata/v2/upsert

The major difference between the two options is that the upsert option offers additional feature that are relevant to simplify integrations. Those features include:

- Automatic handling of creation or update, which simplifies middleware calls to distinguish between a PUT or a POST request
- Allow providing of partial payloads on field and record level
- Allowing also implicit deletion of data using full purge (replace of original data with new data)
- Do not update identical records to avoid that unneeded changes are made to audit records

In general SAP SuccessFactors offers multiple tools to build an integration to write data into EC, the major ones are:

- SAP SuccessFactors Integration Center
- File based scheduled imports in SuccessFactors Provisioning
- SAP Integration Suite

In addition to those there are also multiple 3rd party tools available.

5.4 Managing Dates, Times, Time zones and Daylight saving

Our Employee Central OData Documentation explains here quite well how to toggle on API level between returning the full history of effective dated entities using toDate and fromDate as URL parameters and how to influence via asOfDate URL parameter to get access to an single effective dated record valid at that time. Those are the three major proprietary URL parameters to influence the returned history records of entities.

See also the SAP SuccessFactors HXM Suite OData API Developer Guide for more details on how to query effective dated data. (SAP, 2019)

With respect to the change date there are additional design decisions to be considered. SAP SuccessFactors started storing time information for change dates in server timezone later we added this information also in UTC. (Getting your time zones right, 2019) explains the pattern when a field can be assumed to be stored in UTC and when in server timezone and how this is also visible from OData metadata.

We talked about the different extraction modes: object level, record level and field level delta. Those are also relevant in this context. OData offers of course full load but it does not support field level delta, this is the unique use case for the Compound Employee API. For the two remaining delta modes object level delta and record level delta OData has a special handling defined for last modified queries. In general, when a last modified query without additional filters is being used and the last modified parameter is on the left side of the $filter condition, OData will do an object level delta, e.g. returning all effective dated records of an employee, even if just one has been changed or deleted. As soon as the last modified field is on the right side the result will be a record level delta where only the really changed records will get returned. See also “Querying Effective-Dated Entities Using lastModifiedDateTime” in the “SAP SuccessFactors HXM Suite OData API Developer Guide” on help.sap.com:

https://help.sap.com/viewer/d599f15995d348a1b45ba5603e2aba9b/latest/en-US/4af1754504b04e0d9ccc773bf0b26334.html

5.5 Performing and detecting deletions

In general OData offers the DELETE operation to delete records. While many entities in SuccessFactors support this there are also alternative solutions available such as:

- The full purge option in CSV imports,
- The full purge URL parameter in OData for function import upsert
- The OData operation field in function import

All of those can be used to do hard deletes which remove the records from the original database. As a result, the only information left about this record is stored in the audit tables if being activated.

In order to detect those deletions in data integration scenarios or trigger processes integrations, someone needs a way to retrieve those deletions. Based on the created audit entries the Compound Employee API and the OData Entity APIs can return the information about the deletion. While the Compound Employee API offers different modes to detect those deletions, it is limited to employee master data. OData though can retrieve this information also for other data, such as Foundation Objects or Positions. There are certain limits for OData as well when it comes to complete deletions of objects in a case where even no root object remains to query. This requires a custom solution comparing keys between source and target systems.

Direct access to the audit table via OData or Compound Employee API is not possible nor wanted.

5.6 Accessing pending Data in EC Workflows with APIs

In general, there are two different kind of pending data stores in Employee Central, one for employee central core objects and another one for MDF objects.

At a high-level, the OData APIs for workflows can be divided into four classes:

- Reading basic workflow information's such as steps, participants and comments, including newly created or changed workflows
- Reading workflow attached documents
- Taking action on workflows using OData Function imports for approval, rejection or withdrawal
- Reading pending data

While the first three are the same for MDF and EC Entities the last one has different behavior:

- EC is based on delta store and supports multiple workflows for a record and multiple entities in one workflow (e.g. new hire)
- MDF is based on snapshot (not delta) not supporting multiple workflows for a record and also not multiple entities in one workflow.

As a result, for EC someone must calculate the snapshot based on the current record and the delta. While for MDF the delta must be calculated using the current record and the snapshot.

All MDF and EC based Workflow information can be found in the guides below:

- SAP SuccessFactors Employee Central OData API: Reference Guide – Workflow: https://help.sap.com/viewer/b2b06831c2cb4d5facd1dfde49a7aab5/latest/en-US/1c5fa8b776bd408693400df5e3745498.html?q=Workflow
- SAP SuccessFactors HXM Suite OData API: Developer Guide - Retrieving Workflow Information for Pending Data: https://help.sap.com/viewer/d599f15995d348a1b45ba5603e2aba9b/latest/en-US/2e8574e7ce5f4ec9be996b511bdb5578.html?q=Workflow
- SAP SuccessFactors HXM Suite OData API: Reference Guide - TodoEntryV2: https://help.sap.com/viewer/28bc3c8e3f214ab487ec51b1b8709adc/latest/en-US/cb29d40f60594c559dd1251e084cdcf7.html

Having a use case such as workflow reporting using an API approach this will require to read open workflows and filter those based on the corresponding objects (Position vs. Employee Data) but also based on a set of responsible users (department). In addition, pending data must be read for those workflows. In some cases, we might want to offer navigation to the workflow UI and take action from there or by further API calls.

A 2nd use case was triggering a 3rd party system (e.g. a ticket system), a solution for this will require to create a ticket based on a new created workflow or a certain workflow step becoming active. If this condition is met the ticket shall be created with pending data from the workflow. Depending on the further interaction pattern between the ticket system and the employee central, i.e. if the EC workflow shall wait for the ticketing system to complete by using a workflow step, either further updates on the workflow have to be reflected on the ticket or activities in the ticket have trigger an approval of a step in the workflow.

In general, all workflow-based integration will be inbound integrations since there are no workflow specific intelligent service events. In some cases, there are intelligent service events which are triggered at the same condition as the workflow though, e.g. new hire.

Chapter 5.6 will document the solutions and API examples to build the two scenarios above.

6. DETAILED SOLUTION

All solutions explained here are middleware independent but finally those best practices will have to be applied in a middleware technology. In case the technology of choice is SAP Integration Suite there are additional best practices available. While the corresponding SuccessFactors IDPs are not available yet those are good resources to check before you start building.

All new SAP Integration Suite Design Guidelines are available on the SAP API Business Hub, including content such as example iflows, postman files, etc.: https://api.sap.com/search?searchterm=Design%20Guidelines&tab=all
In addition you will find the complete guide as part of the SAP Integration Suite Documentation on help.sap.com: https://help.sap.com/viewer/368c481cd6954bdfa5d0435479fd4eaf/Cloud/en-US/6803389050a0487ca16d534583414d2b.html?q=Guideline

In addition to the SAP Integration Suite best practices above there is also a “OData API Best Practices” summary available as part of the “SAP SuccessFactors HXM Suite OData API: Developer Guide”:
https://help.sap.com/viewer/d599f15995d348a1b45ba5603e2aba9b/latest/en-US/da51d32dc17945749333cfb4e22f2bf4.html

6.1. Using the right APIs

When building new custom integrations there are multiple options to use either the SOAP-based Compound Employee API, OData entities or file-based CSV uploads. Since SAP does not recommend building file-based integrations if they can be avoided, below we documented some of the best practices in which scenario what kind of API or combination of APIs shall be used. The graph illustrates high level which integration requirements result in corresponding recommended API usage.

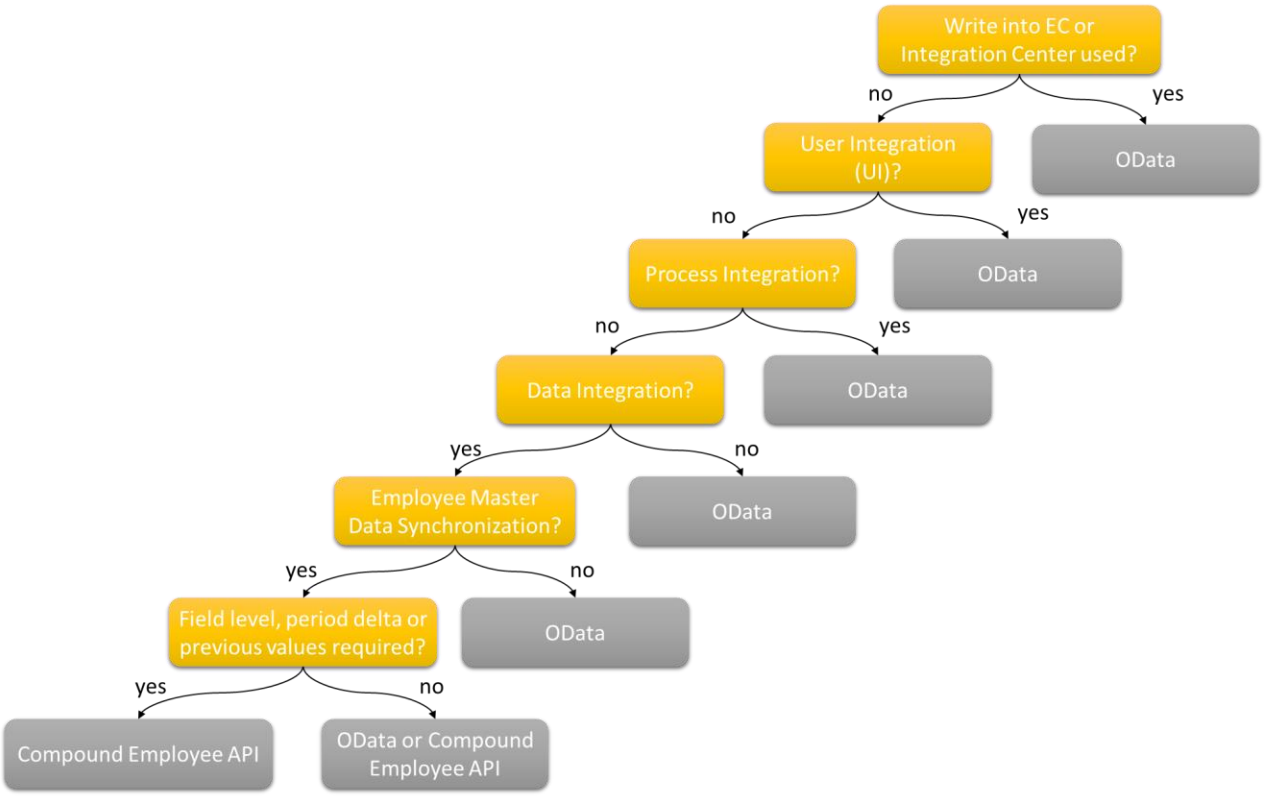


Figure 1: Decision tree for Compound Employee SOAP API or OData?

In the case were both APIs can be used, we recommend to do a feature and a performance comparison based on the concrete customer data to ensure that work required to enrich data from the Compound Employee API with OData (e.g. labels, descriptions from Picklists or Foundation objects) is in balance with the performance improvements compared to the pure OData API usage. The Table 1 below shall help to take this decision.

From internal performance number we saw that for the overall execution time, when comparing an object level delta of OData with Compound Employee SOAP API, the difference between the two can be as much as a factor of 3. The highly optimized but limited Compound Employee API is usually faster than the generic OData API. Of cause this factor changes if further data must be read in case of CE API or if filter conditions are not available. Therefore, in cases where both are valid to be used, a case by case decision and a comparison is important.

Even if the right API is chosen, the next questions is how to use this API in the most efficient way to ensure, robustness, high performance and security. The following paragraphs will address this question.

Capability	SAP SuccessFactors Employee Central OData API	SAP SuccessFactors Compound Employee API	SAP SuccessFactors Employee Central CSV File Import
Query – Field level delta	No	Yes	n/a
Query – Previous values for field level delta	No	Yes	n/a
Query – Period Delta	No	Yes	n/a
Query – Record Level Delta	Yes	Yes	n/a

Full User based field, entity and target level Permission for read	Yes	No	n/a
Simple Target Population based Permission for read	No	Yes	n/a
Deep Filter	Yes	Yes	No
Complex Filter	Yes	No	No
Simple Filter	Yes	Yes	No
Retrieve inactive User data (e.g. after employees are purged)	Yes	No	n/a
Retrieve non-EC Employees (Users/Employee Profile)	Yes	No	n/a
Retrieve all person types	Yes	No	n/a
Session Reuse	Yes	Yes	n/a
Cursor based paging	Yes	No	n/a
Snapshot paging	Yes	Yes	n/a
All EC Person and Employment Entities	Yes	Yes	Yes
All EC Entities supported incl. Foundation Objects	Yes	No	Yes
User based MDF Objects	Yes	Yes	Yes
Background Elements	Yes	No	Yes
Non user-based MDF objects	Yes	No	Yes
Use for Data Integration	Yes	Yes	Yes
Use for Process Integration	Yes	No	No
Use for User Integration	Yes	No	No
Use for IoT Integration	Yes	No	No
High data volume	Yes	Yes	Yes
High frequency calls	Yes	No	No
Can be used in Integration Center	Yes	No	No
Write Data	Yes	No	Yes
Trigger rules	Yes	No	Yes
Trigger Workflows	Yes	No	Yes
Trigger IS Events	Yes	No	Yes

Table 1: Feature Comparison Compound Employee API, OData and CSV based Import

One of the most important points about choosing the right API is to choose the latest version in case there are multiple versions available. While this is easy to figure out in case the Entity or URL contains the version number it becomes more difficult if the version includes a complete technology shift as this was the case with SOAP. As of today, except for Compound Employee API, all Employee Central SOAP entities shall not be used anymore and instead the OData APIs shall be used, this deprecation includes also AdHoc SOAP APIs.

6.1.1 Keep the result set small – Data retrieval in delta mode

In order to improve the overall performance of the integration there are certain patterns to be followed. The most important one is to keep the result set of transferred data small. This shall can be achieved with the SF APIs by using a delta mode instead of full loads, e.g. using object, record level or field level delta loads to transfer only changed data of entities or business objects instead of all data in the database for those.

The Compound Employee API supports all extraction patterns full loads, object level, record level and field level delta transmission. The CE API talks about “full transmission mode” when the extraction pattern is objects level delta and about delta transmission in case of record or field level delta:

- In full transmission mode, the API replicates the complete employee data including future and historical data, considering only the employee data that was changed during the scheduled run.
- In delta transmission mode, the API only returns employee data that was created, changed, or deleted since the last replication:
    - Effective-dated delta transmission is designed for consumers that work in an effective-dated manner. This means, it is assumed that consumers store the time frame (start and end date) when a data record is effective. Deltas are communicated with regard to the time frames that have been transmitted in the last replication.
    - Period-based delta transmission is intended for consumers that are not able to deal with effective dated objects. Deltas contain retroactive changes and effective dated objects that are relevant for the given period

On the other hand, for the ODATA APIs there are two ways to query effective dated entities:

- asOfDate query: returns information as of a specific date. Always returns a single record. If omitted, OData API implicitly includes asOfDate to be today.
- fromDate and toDate query: returns historic information for the specified date range. Can return multiple records, depending on information available. For example, if the fromDate is 2019-03-15 and the toDate is 2019-3-30, the returned effective data would be like effective_start_date<=2019-3-30 and effective_end_date>=2019-3-15. Query request: /jobInfo? fromDate=2019-3-15&toDate=2019-3-30 will return records from 2019-3-15 to 2019-3-30 for jobInfo entity.
- Using last-modified queries together with expand allows also OData to retrieve record level delta information for hierarchical object

This blog explains the three different delta modes: https://blogs.sap.com/2017/09/17/api-data-extraction-patterns-from-successfactors-employee-central-for-delta-interfaces/

One key benefit of the Compound Employee API is the specialty of gathering a full picture of all the employee data (all the portlets) with only one call – which saves a lot of time for the development of the integrations, compared to the REST OData calls where more effort is needed in order to join all the entities. In general, it has been found that Compound Employee API has better performance than ODATA API.

Even if the receiving system is not capable of retrieving delta information, custom integrations should aim to implement a delta pattern, especially if the frequency of the synchronizations is high or the amount of transferred data is big. In case there are many systems which are not able to retrieve delta information it should be best practice to keep the last version of the full picture close to the receiving system and update this with delta information before uploading instead of running several full loads at the same time.

In case of master data synchronization, it is also good practice to be able to do full loads in case of exceptional situations, e.g.

To fix missed delta integrations

- Do an initial load
- Fatal errors (recovery from backup)
- Long term bug in the delta integration and unclarity for how long
- Delta integration wasn't running, and nobody knows anymore when the last successful run was
- If there was no replication for more than 3 months

Those full loads shall always be an exception and not a default pattern.

6.1.2 Keep the result set small – Usage of filters

While the Compound Employee APIs offers only a limited set of filters the OData APIs provide a much broader set of filters. In addition to the default time-based filter mentioned above to limit the amount of retrieved history information and reading only changed data sets (field and record level delta) another powerful way to reduce the amount of transferred data is to use additional filters. In case only special employees are required (country, employee type, status etc.) those filters shall be used to limit the employees leaving the system instead of filtering them out in the middleware or in the receiving system. OData offers also one powerful capability called a deep filter. Filters can be defined not only on the root entity but also on deeper entities. Even though the OData query starts with PerPerson this allows to filter for Employees who have a Job Information record with a special department. The example below shows such a request to get the first and last name of all persons which are as of today in the department with external code 5000133:

GET https://apisalesdemo4.successfactors.com/odata/v2/PerPerson
?$filter=employmentNav/jobInfoNav/department+eq+'5000133'
&$format=JSON&$expand=personalInfoNav
&$select=personalInfoNav/firstName,personalInfoNav/lastName

While this result can also be achieved starting from EmpJob, since navigations are offered also in the opposite direction, it is not possible anymore when those navigations are missing. E.g. to get all employees which are as of today in a department with department name 'Production':

GET https://apisalesdemo4.successfactors.com/odata/v2/PerPerson
?$filter=employmentNav/jobInfoNav/departmentNav/name+eq+'Production'
&$format=JSON
&$expand=personalInfoNav
&$select=personalInfoNav/firstName,personalInfoNav/lastName

The query above is only possible in a single query like this. Without this deep filter the only option would be several queries a first one to read all departments with this name and a 2nd one to read all employees with the department code with the query above using an IN clause instead of EQ since we now will get multiple external codes from the department query. In case there are many external codes we even must split the PerPerson query since IN filters are limited to 1000 entries (or less when URL length is the limit and $batch is not used).

The best way to retrieve data is of course to read all of it with one API call and use paging to retrieve the results instead of multiple API calls. In case this is not possible, e.g. someone has only business keys of objects to be retrieved and requires not detailed data, this can happen for example if the Compound Employee API is used and additional information about Foundation Objects is required, a good practice is to retrieve this information using a IN clause in the corresponding filter condition of the OData API request. This IN-clause is limited to 1000 entries and is more efficient than single API calls. If the URL length limitation is reached, we recommend limiting the number of parameters. Use $batch to work around this limitation only as a 2nd option.

Even though this is part of the best practices for building integration in SAP Business Technology Platform or any other middleware, we also like to recommend in this document to make use of the caching mechanism in the middleware in case data has to be enriched several times, e.g. picklist, foundation objects, to avoid multiple APIs calls with similar or the same data during processing.

6.1.3 Keep the result set small – Limit selected fields and joins using $select and $expand

While the Compound Employee API only allows to limit entities (only in delta mode we can make use of query parameter changedFieldsOnly), OData allows in addition also to limit the returned data on a field level using $select. A best practice is to make use of $select to limit the number of fields in case they are not required. This helps to reduce the amount of data and keeps the results set stable in case additional fields are added. On the other side it makes the APIs call less robust on field changes, e.g. if fields are removed from the data model, API calls listing a removed field in $select will fail, but removing API field from a productive system shall anyhow be avoided.
What is true for $select, is even more true for $expand. $expand shall be used in OData but someone should expand entities only if those fields are needed, use again $select to limit returned fields of expanded entities.
If $select and $expand is used often, the URL length might reach the critical limit of more than 2047 characters for some clients (such as internet explorer) and up to 8k for our internal servers. In such a case the OData API call can be put into $batch as explained in chapter 5.2.
In addition to $select and $expand the returned response can also be compressed using a header parameter called Accept-Encoding wit value gzip. This helps to reduce the message size significantly especially in case of ATOM format.

6.1.4 Save compute time – read User instead of EC Structures

In order to avoid wrong usage of data from the User entity in integrations, SAP recommends to build integrations in general on the Person and Employment OData entities to ensure that they make the right use of those two different concepts: the Person and the Employment . Otherwise those integrations might fail in case EC is introduced and multiple userIds exist in User Entity for each Person due to Multiple Employments caused by Global Assignment, Concurrent Employments or International Transfers on new Employments.

In case only as of date information is needed or a mixed EC/non-EC scenario exists and all the data needed is part of the OData User entity someone could use this entity with care instead of joining the EC-Structures in a complex OData call. In such a case ensure that multiple Employments will not become an issue when EC is being implemented,
Note: Data structures for the OData User entity and Employee Central entities differ. Please inspect the entities to ensure that you get the data needed, from one or the other.

Different than the EC Entities, the user entity will be updated in a delayed way from the EC entities using HRIS Sync. Hence this delay must be considered too. In case EC is not used, the User entity is the only entity to get the employee data from.

6.1.5 Save compute time – authentication and session reuse:

The Odata APIs allow two types of authentication:

- HTTP Basic Authentication: it is less secure but simple to use, recommended for testing purposes and it is based on the Username, Company ID and Password. It can only be user if SSO is not enabled in your SuccessFactors instance (a possible workaround would be to change from Full SSO to Partial SSO). See also https://apps.support.sap.com/sap/support/knowledge/public/en/2505553. This problem can also be addressed by setting API login exceptions for the API users.
- Oauth 2.0: it is the most secure and recommended option for the custom integrations. It is based on the registration of the Oauth client. After the registration is done, a X.509 certificate will be generated which needs to be used in your application (the private key of it) in order to make the token requests for authentication.

From a security point of view someone should always use oAuth instead of Basic Authentication. The only use cases where basic authentication makes sense is development or testing and of cause if the client doesn't support oAuth. If being used for testing, please ensure that no similar usernames and passwords are being used as in production.

Before 1911 oAuth could be used to get an access token for any user in the system using a SAML assertion. With 1911 the oAuth client also supports the concept of a technical user which allows to bind one single technical users to the oAuth client. In this case the user cannot be changed anymore in the oauth/idp call but is fixed to the defined technical user.

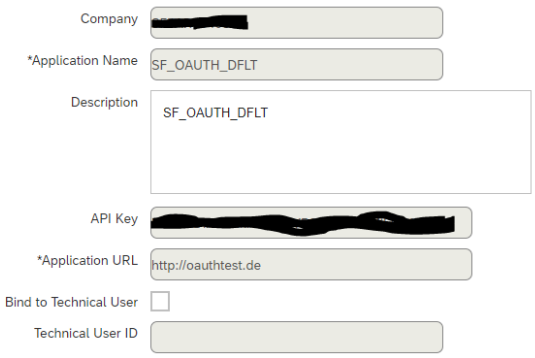


Figure 2: oAuth2 application client with option to bind to a technical user

For the CompoundEmployee API, the authentication is established through the Login operation. A successful login will return a session ID as an HTTP Cookie. This cookie must be passed back to all subsequent HTTP Requests that invoke API operations in order to authenticate. The API session will timeout after 10 minutes of inactivity. You can also manually invalidate a session using the Logout method.

Especially when calling the APIs from a middleware it is important to reuse the login sessions of OData API calls. In case of SAP Integration Suite this must be configured on iFlow level (Runtime Configuraiton) and not in the SuccessFactors Connector.

It is important to know that session reuse and multi-threading will only work in case the different threads use different sessions. Otherwise all threads will hit the same API server due to sessions stickiness. SAP will throttle such request in future. Always stay below 10 threads. For less than 50k records multithreading won't improve performance much and is not recommended.

Except for special cases we always recommend using the SuccessFactors connectors instead of the generic OData or HTTP connectors in SAP Integration Suite.

6.1.6 Save compute time – Role Based Permissions and Read Access Logging

For the cases when the integration needs a logic based on role based permissions, the recommended option is to use the ODATA APIs. There are two modes which determine what a user or user group is authorized to view or do.

- User Mode: Assigned RBP settings determine what entities can be viewed and what can be done with them.
- Admin Mode: Allows full access to OData API entities and operations. You will only use this in a limited number of cases chiefly technical integrations. Admin Mode overrides any RBP (role based permission) settings that have been made.

User Mode permission are the only secure option when building an application UI on top of SAP SuccessFactors APIs using OData APIs and oAuth.

While process integrations might require that data being created is processed using the named user (instead of technical API users) including full permissions checks, this is usually not the case for data integrations. Master data synchronizations usually are executed with technical users. Avoiding permissions checks in this case is a huge performance saver. Using Admin mode for master data synchronization should always be considered as the default option. Only in cases where legal or corporate regulation enforce a separation of

data also on technical user level, when replicating data into multiple target systems for different subsidiaries or companies with a holding, this approach can be used.

Different than user level permissions checks, it is a good practice to create several API users and oAuth clients for different data integrations. This allows the customer to differentiate the different integrations also by the API user but in addition limit the authorization of this API user only to those APIs which are required, e.g. a master data replication integration and the used technical user do not require write permission in Employee Central. By this the customer can define a scope of the corresponding integration without reducing the performance by adding record or field level permission checks.

Read access logs should be disabled for technical users in general for performance reasons. This can be done by defining an exception for the corresponding API user in the settings for the read audit:
https://help.sap.com/viewer/2becac773fcf4f84a993f0556160d3de/latest/en-US/efa0d6838b6c4b3d94f289dca50aed64.html

6.1.7 Outbound communication (events)

SAP SuccessFactors offers Intelligent Service based on Events. See “Setting Up Intelligent Services – Intelligent Services at:
https://help.sap.com/viewer/e85fac4e6a3b4884b6451d4208cdb778/latest/en-US/2771a2e7e6c14ef48e3f9ba91491a957.html

Those events can be used to trigger a SOAP listener outside of SuccessFactors. They are defined in SuccessFactors and triggered by business rules in case of Employee Central. SAP SuccessFactors recommends using those events to trigger real-time integrations to other systems for process integrations instead of scheduling integrations close to real-time (fast polling), e.g. replicate a new hire immediately or create a ticket in a ticket solution in case a new position is created. It should not be used to build an event driven data integration or master data synchronization.

It is important to mention that someone should not change the semantic of the trigger of an event, e.g. misuse a new hire event to trigger it in a different context, since other listeners might rely on the semantic and context of this event.

There are two major risks when it comes to events. Different than scheduled integrations they are much more difficult to track and control and someone needs for initial loads or case where the receiver could not process the events a mechanism to transfer data also without them. Beside not getting the events, due to longer downtimes of the target system etc. there is also the risk to overload the system with too many events. Especially in case of mass processing this might be a risk. In a worst case this might even result in an endless event loop in case the event triggers an integration which in turn triggers again the same event.

This blog explains an additional flavor of using a code free integration based on Events using the Integration Center. The Event triggers in this case the integration built in the integration center in order to call a REST endpoint:
https://blogs.sap.com/2018/04/23/successsfactors-intelligent-services-center-isc-integration-center-ic-replacement-of-3rd-party-middleware-to-trigger-simple-interfaces-from-employee-central/

While event-based communication is not available yet for all use cases it is also not an option to simulate this by high frequency API calls.

SAP does not recommend to schedule delta queries below one hour to simulate a real-time behavior for data integrations because this has significant negative impact on resource consumptions and performance for other calls

---

6.1.8 Usage of the right pagination

For the Compound Employee API, server-based paging as mentioned in the API documentation is the most efficient version. This is the efficient version only when you have many employees. If you have only some employees (example 50), then the most effective way is to have no paging. For OData there are three different

paging options: client-based paging, server-based cursor and server-based snapshot. While the client-based paging is only suitable for UIs (user integration scenario) the server-based paging versions are designed for data and process integration styles. Server based paging is the only approach where missing records in delta replications can be avoided. According to the OData API Developer Guide Snapshot based paging can improve the performance of complex queries but also comes with some additional risks of timeouts since the first page has to retrieve the complete set of keys.
In general SAP recommends to use snapshot based pagination if your integration scenario fits into the limits of the approach:
https://help.sap.com/viewer/d599f15995d348a1b45ba5603e2aba9b/latest/en-US/7af42e7079c3481688de975b38867c41.html
If this limit is reached split the query into multiple ones. One way to achieve this is for example to do a $count on the corresponding root entity and use a unique increasing key value, e.g. userId or personId, to split the query into 2 or multiple equally long parts.

A common pitfall in API calls is to use client-based pagination without any order. In such a case the result set is random, and each page might contain duplicates or records are not returned at all. When using client sided paging $orderBy must be used to create meaningful results in user integration styles. Please see chapter 8.1 for those.
Another important consideration while using paging is object boundaries in case of effective dating. Paging is not aware of those object boundaries, as a result an object such as a position can span two pages. This might happen if the object was part of the first page, but subsequent effective dated records are part of the next page. While this is not a problem in general, it might become a problem in case pages are processes one after the other to create the objects, in this case the position, in a target system. This packaging of messages by API pages is a typical pattern used in the middleware to avoid out of memory exceptions. As a result of a per page processing the object will get created partially from the first page and additional records will get created with the next page. This two-step processing of the object might result in unwanted side effects or even in errors in case the partial object cannot be written. To avoid this there are several strategies

- Create a file and process the file later on to create the objects respecting the boundaries, such a file can be created using the integration center
- Use a staging area in the target system which is filled and processed per object after the last page is filled
- Use an own packaging algorithm in the middleware to avoid splitting up objects by creating new messages using a splitter and split the pages of the API call into messages respecting the object boundaries

6.1.9 Save compute time – Suppress triggering of rules

Note: This section is particularly applicable in case of imports or writing data to SAP SuccessFactors via API calls.

Most entities in Employee Central trigger rules, workflows and IS events (Intelligent Service Events, ISE) in case they are changed. Usually this is wanted in case this data is being changed or created from the UI and sometimes even if this data is being changed or created by an API call. But in some cases, those rules shall not be triggered, for example if data is provided already in the right way and rule processing is not required anymore. In such a case there are several options to suppress rules.

- Make use of the permission settings for Employee Central imports to disable triggering of rules for certain entities. To avoid side-effects, use a dedicated “technical user 2” for this. This will allow you to have some integrations triggering rules using a “technical user 1” while others using “technical user 2” will not trigger rules
- Make use of the rule context in the business configuration UI to trigger rules only in case of UI but not in case of API calls or imports
- Disable rules from the business configuration UI, while you are doing a data migration or initial data load
- Use a special hidden field in the object to control when rules are triggered and when not.

6.1.10 API Documentation and Discovery

The best place to discover our OData APIs is the SAP API Business Hub http://api.sap.com. This acts as a central entry point for our APIs and packaged integrations. The available feature includes:

- **Discover**: after an initial search the result can be filtered by Product, e.g. SAP SuccessFactors and the artifact type, e.g. API or Integration Package (for SAP Integration Suite)
- **Overview**: Short introduction of available operations and structure of responses
- **Download**: Get a Swagger specification of the API for the request and response
- **Documentation**: Links to API documentation from the detail screen of each API
- **Testing**: Testing of APIs either with a real system or a mock data system
- **Code**: Generation of code snippets for Java, Swift, Curl, SAPUI5 or ABAP

Note: The SAP API Business Hub does not list all SuccessFactors APIs, SOAP APIs including the Compound Employee and most deprecated APIs are missing. The tentative roadmap plan for Compound Employee API to be made available on SAP API Business Hub is Q2 2020(b2005 release). You also won't see all available integrations in the hub, only SAP Integration Suite based ones. A limited number of Integration Center templates are also available.

Also note that as part of the SuccessFactors API Center we also offer built in capabilities in the product to discover APIs. Different than the SAP API Business Hub this tool has less features for discovery and test but shows the available APIs and their metadata which differs for each tenant based on the configuration and enabled modules and features.

One nice feature of the OData APIs is the technical flexibility of retrieving the full data model (metadata) configured a specific SuccessFactors instance with one single API call, while this is not available with the Compound Employee API for all entities. This comes in very handy for the cases when you don't have direct access to your SuccessFactors instance in order to check directly the API Data Dictionary.
In order to retrieve the metadata, the integration consultant needs to perform the following call: https://<your_sf_domaini>/odata/v2/$metadata
After this the corresponding file can be viewed offline or even imported in Microsoft Excel for better viewing.

Note: This $metadata API can also be used in any integration to dynamically react on the current configuration of the data model of the SuccessFactors tenant. This is also a good practice to avoid APIs failing in case there are data model changes.

All SuccessFactors API Documentation can be found in the Development section on help.sap.com for download: https://help.sap.com/viewer/product/SAP_SUCCESSFACTORS_HXM_SUITE

6.1.11 Communities, Support and Feature request

The major community for SuccessFactors APIs is "API and Integration forum" in the SAP SuccessFactors Partner Community:
https://partnercommunity.successfactors.com/t5/API-and-Integration/bd-p/API-and-Integration
This community is the one stop for all SuccessFactors related partner questions. Another community worth following is the SAP Community and Blogs: https://blogs.sap.com/

Product issues are reported using our normal channels: support.sap.com using the component for the corresponding API. In case this is not clear use LOD-SF-INT-ODATA or in case you know the module LOD-SF-<module>-API.

API feature request similar as any other product request are filed via the SAP influence platform: https://influence.sap.com/

New API features will be mentioned in the release webinars first, followed by a publication in the corresponding API guidelines and the SAP API Business Hub. If new APIs are being enabled (some APIs have configurations switches to enable them) they will show up with the release update in $metadata.

Find more details about release webinars and roadmap see here:
https://community.successfactors.com/t5/Product-Updates-Blog/bg-p/ProductUpdates

6.1.12 Common Pitfalls

There are some common pitfalls when using APIs we like to high-light here.

Avoid incompatible API changes caused by data model changes.

See "Broken APIs: What causes them?" on help.sap.com for details.

Don't forget to refresh metadata after data model changes, even though this is done automatically in many cases. You can use the API Center UI for this or the OData API call.

Implement proper retry mechanism for those http error codes where this makes sense:

- KBA: https://apps.support.sap.com/sap/support/knowledge/public/en/2735876
- Help:https://help.sap.com/viewer/d599f15995d348a1b45ba5603e2aba9b/latest/en-US/da51d32dc17945749333cfb4e22f2bf4.html

Please take into account that it is also a good practice to use instead of a constant retry pattern a non-linear and/or random retry pattern to avoid that in case of high load a retry of many failed processes at the same time makes the situation even worse. Retries should be limited to 5 as well. Patterns could be 2,4,8,16,32 Minutes with some additional random delay. A retry usually makes sense in case of http error code 5xx different than 500. For details when a retry makes sense and when not, look also here: (SAP, 2020)

Know the limits of APIs and build your processes accordingly or react dynamically if those are reached
In the appendix in chapter 8.1 we collected a list of limitations coming from using http/https as a communication protocol or boundary conditions to safe guard infrastructure. Those include size and time related limitations but also limitations with respect to conditions.

Set the password of your API user in a way that it does not expire. If a password of an API user expires the integrations will start failing. This can be avoided by defining an exception in the "Password and Login Policy Settings" in Admin Center. Also be aware that certificates for oAuth usually expire after a few years and have to be replaced in time.

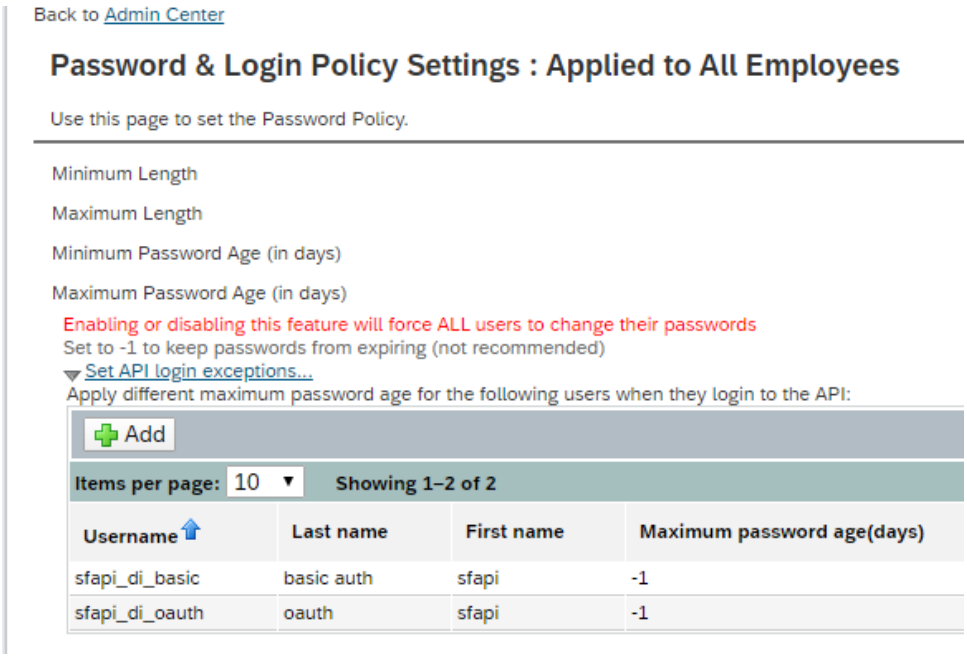


Figure 3: Adding an exception for the password expiration for technical users (API)

SAP SuccessFactors offers public APIs for consumption. In addition, there are internal/restricted APIs and also beta APIs. Both of those categories shall never be used in a productive environment. In addition, there are also deprecated APIs, which are clearly marked as those in our tools and the documentation. In general, we explain our deprecation process and implication here:
https://help.sap.com/viewer/1a981b4253d849eb82d4ca32bc767750/latest/en-US

Another common pitfall is the usage of unstable properties aka changeable IDs such as personIdExternal, username, email, in some cases even userId. Use instead personGuid, assignmentIDExternal or personId. Using those stable IDs will allow you to build reliable integrations, while you might loose track of changes in data replication scenarios when those changeable IDs are used and changed.

There are multiple APIs which cause again and again issues due to wrong usage. Those include for SOAP Compound Employee and legacy AdHoc APIs (for new implementations AdHoc APIs must not be used anymore) and OData Picklist, Attachment, snapshot pagination. In the interest of stable and performant integrations, please monitor those integrations in specific and check for the right usage of those APIs in the former chapter. In case you see a decrease in the performance of API calls it is recommended to either split the queries by reducing complexity or splitting the results set by having multiple calls. All other before mentioned recommendations to reduce compute time or keeping the result set small shall help as well.

6.1.13 Secure API calls

As mentioned already in the permission chapter it is best practice to use oAuth and technical users also for integrations instead of basic authorization. For data integrations named users or admin users with login permission in the system shall be avoided.

Having dedicated users for each integration adds additional security and avoids that APIs users have always access to all APIs. This allows to have a kind of access scope for integrations and API users.

While it is a best practice to define white or blacklist for IPs, this also comes with some burden to adjust in case the landscape is reorganized. Changing the middleware for example from Dell Boomi to SAP Integration Suite or changing Data Centers creates the need to adjust those filters to ensure that integrations still work.

6.1.14 Improved Monitoring and Support

For better Monitoring and Tracing SAP recommends to used the following headers for each integration:

- **X-SFSF-Tenant** for the tenant or the SuccessFactors company ID you are calling to
- **X-SFSF-Process-Name** for the iFlow or process name calling the APIs
- **X-SFSF-Execution-Id** for the execution ID of the process calling the APIs, e.g. UUID

By using those parameters customers can better find APIs calls belonging to each other in the corresponding APIs logs and SAP support will also be faster to extract application logs from our internal tools.

6.1.15 Deployment

While customer usually have only limited influence on the deployment of SAP internal cloud products, they and their partners have control on what kind of infrastructure they are using to build integrations in case there are multiple options available. We always recommend to use backends, middlewares, SFTP severs and databases which are close to each other. In case the customer deploys additional optional components used in integrations such as own middlewares or SFTP servers those should be collocated or closer to other components in case the data volume is high.

- Try to avoid using SFTP server in other DCs which are not close to the backend systems using or writing this data, e.g. do not use SFTP server coming with other products such as Hybris and which are located in others DCs with SuccessFactors
- Try to avoid having your middleware in a different DC than all your connected systems

6.2 Batching SuccessFactors OData calls

6.2.1 Calling complex OData APIs

It is important to mention that before moving the URL into a $batch statement someone shall try to reduce the URL length accordingly. Only if this is not possible, or not possible without reducing significantly the performance of the integration using the API, we should move the API call into $batch.

Even though there is no clear limit when a OData call is too complex, SAP recommends checking the performance of OData API call while increasing the complexity. Complexity will get increased when new joins are added using $expand and the number of conditions in $filter are increased to create delta queries across the joins. Moving complex OData queries which exceed the URL length into $batch makes only sense if the performance is still good. If the performance decreases in nonlinear way when the complexity is increased linear it is a good idea to split the request into several request and put those into $batch.

As a rule of thumb those are complexities in OData which should get avoided and are at least a reason to monitor the performance of the API call in order to split it and reduce the complexity if required:
The depth of the OData API call exceeds 4 levels ($expands contains paths with more than 3 slashes)

- Deep filters are used below level 2, e.g. $filter=employmenNav/empJobNav/location+eq+…
- Multiple deep filters across one-to-many relationships
- More than 20 filter conditions
- Filter values longer than 100 bytes
- Number of expanded entities above 10

6.2.2 Improving Performance

Another good use case for $batch is avoiding roundtrips and improving performance. By avoiding multiple single APIs calls and putting them into a single $batch call, the overhead of each call is reduced.

While batching API calls does improve the performance by saving roundtrip time and overhead, it does not improve the performance much on a DB level or business processing level. At the end the corresponding SQL statement will be executed for each API in $batch. Hence it is important to mention that if mass operations are available from the API, that those shall be used, e.g. IN clause for OData $filter or $expand in OData or also deep filters in EC OData APIs. Those will improve both the performance for network communication as well as the backend execution time. $batch improves performance, especially response time much more in user integration styles rather than the throughput in data integrations styles.

Combining API calls must be done in a way that the complexity and performance of the API is in balance. In general, someone should try to join all data in the backend instead of doing an in-memory join in the middleware. In case this in-memory join cannot be avoided ensure that you are batching your messages in the middleware to avoid out of memory situations and reduce the batch and/or page size if required. See also https://blogs.sap.com/2017/08/22/handling-large-data-with-sap-cloud-platform-integration-odata-v2-adapter/. We will have a more detailed guidance on paging and batching in SAP Integration Suite in one of the upcoming IDPs on building custom integrations on SAP Integration Suite.

While $batch can improve performance of API calls it can also lead to timeouts in case too many calls are bundled in one $batch, overloading a $batch statement to work around the paging of the APIs, e.g. include several API query calls with maximum page size is not recommended. We do not recommend using more than one paged OData API call in $batch. The only use case for a paged OData API call in $batch should be the URL length limitation, multiple single record API calls without paging or transactional behavior using changesets.

A positive side effect of Batching API calls is also less data in the API logs which makes tracking of API calls easier. But also hides Entities being used.

6.2.3 Creating Custom APIs

While $batch can be used to combine API calls into one, there is not much logic which can be added. In case a new simplified API shall be created using existing SF OData APIs it is a better approach to use either SAP API Management together with SAP Business Technology Platform or SAP Integration Suite to define this new API with additional logic.

https://blogs.sap.com/2017/12/13/sap-api-management-api-implementation-and-management-on-cloud-foundry-part-1/
https://answers.sap.com/questions/640938/custom-api-code-in-sap-api-management.html

Please not that SAP API Management and in some cases also SAP Integration Suite might require an additional license of the customer. SAP API Management internally used by SAP cannot be used by customers or partners.

6.2.4 Security

Putting OData API calls into a $batch statement also allows to ensure that URLs containing sensitive data are not visible in some of the API logs tracking those parameters, this is of course not an additional security if the payload is logged as well. From an https point if view this does not add any additional security since the protocol encrypts both the URL and the payload, only the domain part of the URL, but not the path behind that, might be visible. See also here.

6.3 Writing data into Employee Central (EC)

Similar as for reading data it is the recommendation of SAP also for writing data to use simpler tools such as the SAP SuccessFactors Integration Center first before starting a complex integration on a middleware. In case this is not possible, due to complexity, security or architecture reasons the only option left is an API based approach.

While most integration processes read data from Employee Central and write data into other systems in some cases customers operate Employee Central as a slave system where the master system is another non-SAP HR System.

SAP SuccessFactors offered in the past SOAP and OData APIs to write into EC (beside CSV based manual or schedule SFTP import). As of today, the SOAP based write APIs are deprecated and only the OData based APIs shall be used. OData in addition is coming with two major operations to write data PUT/POST on each Entity but also the function import UPSERT. While PUT and POST are often used in simple user integrations contexts the upsert operation is the recommendation for all integration based approaches.

6.3.1 Ensure transactional behavior

In case performance is not the most important requirement but transactional behavior is wanted, the usage of $batch and changesets in combination with upsert is the best option to go. In this case the caller of the OData $batch API has the full control over the transactional behavior he can decide which records of which entities belong together and shall behave as a unit. This means either all data within this unit is being written or it is rolled back in case any of those records throws an error.

Those blogs provide good examples on how to use $batch and changesets to achieve transactional behavior and create a new hire API for employee central:

https://blogs.sap.com/2019/12/19/creation-of-an-employee-in-sap-successfactors-using-batch-api/
https://blogs.sap.com/2019/12/18/update-employees-records-from-different-entities-using-odata-batch-api-in-sap-successfactors/

In addition to the above blogs with API examples, there is also a good SAP productized integration making use of this approach. It is the „Packaged Integration - SAP Fieldglass to SAP SuccessFactors Employee Center“:
https://api.sap.com/integrationflow/Packaged_Integration_FieldGlass_to_SF_EC_-_Contingent_Worker

It is important to note here that transactional behavior per employee comes with additional cost and is hence much slower than upserts which are updating data for hundreds of employees at the same time without transactional behavior. In case of migrations or initial load where throughput is the most important factor, writing data without transactional behavior is the better option. Some of SAP packaged integrations, for example the “ERP to SAP SuccessFactors Employee Central Employee and Organizational Data” use this approach, see groovy script use by this iFlow:
https://api.sap.com/integrationflow/com.sap.PA_SE_IN.erp2ec.SAPtoSFSFGenericODataUpsert.v1/resources/script/generate_AtomXML_Request.gsh

6.3.2 Triggering Business Rules via APIs

Only some objects in Employee Central support triggering of business rules from OData APIs and Imports:

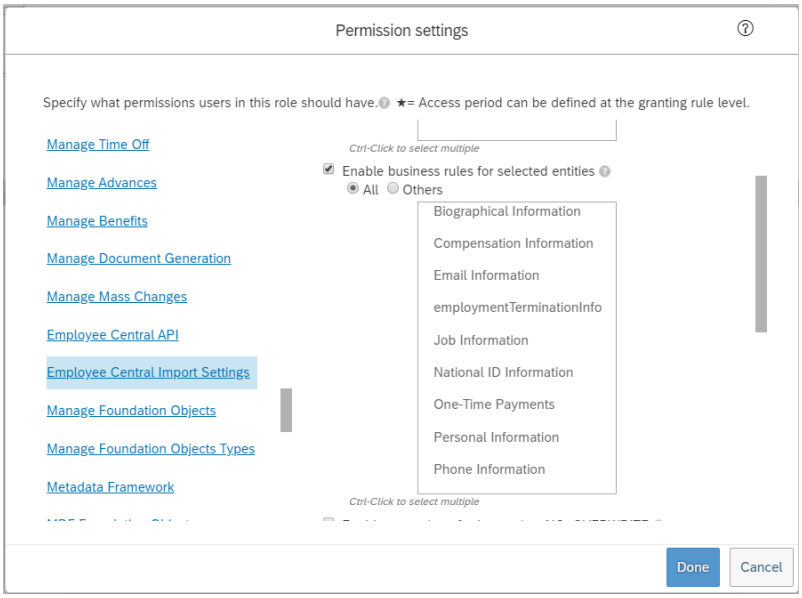


Figure 4: Permission settings to enable triggering of business rules for selected entities

If the above settings are made for the API user, the rules are triggered (no matter if on_change, on_save or on_post_save rules).
Having this setting per user allows partners to have different kind of API users, some enabled for business rules and others without. This allows someone to build integrations which do not execute rules for example in the context of data migration (initial load) or data integration where data is coming with the right data and no additional business rules shall get triggered (and hence also no workflows or events).

Assuming that the above setting is made, there is another so called rule context available the “Manage Business Configuration UI” this rule context has to be set to enabled for the context of imports but does not affect the API behavior.

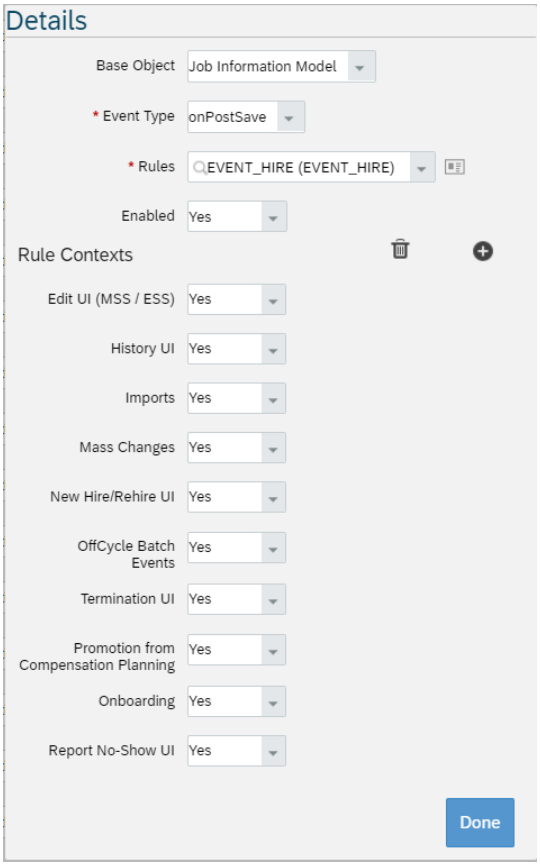


Figure 5: Rule Context defines in which context business rules are executed

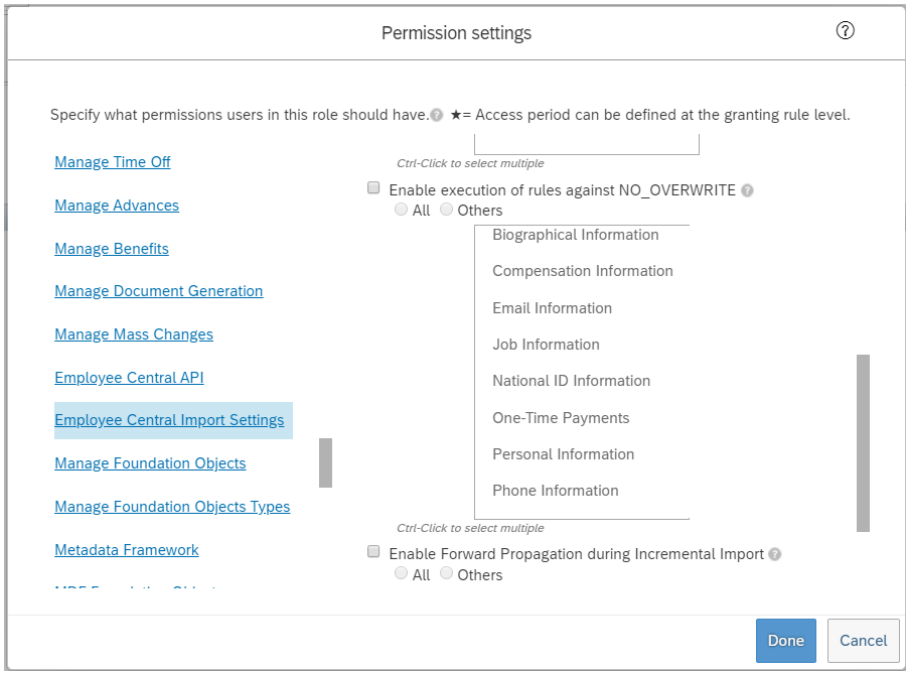


Figure 6: This setting influences the write behavior for fields not specified in the OData API request. If this setting is enabled rules or forward propagation are allowed to fill the field.

In case of writing into MDF, only the on_save and on_post_save rules are triggered but not the on_change rules. There are also no specific RBP setting to enable triggering of rules per user. See also this KBA: https://apps.support.sap.com/sap/support/knowledge/public/en/2173085

6.3.3 Forward Propagation

Forward propagation can be enabled for effective dated entities such as Job Information, Compensation and PayComponentRecurring. In case this setting show in Figure 7 is enabled all import and write API calls for those entities will forward propagate data as explain in chapter “Forward Propagation in Employee Central Imports” in the imports guide:
https://help.sap.com/viewer/f5c753ba58814ef0ab181747824c41ed/latest/en-US/2f4aef17cedc4e80bd492accbb662b35.html?q=%22forward%20propagation%22

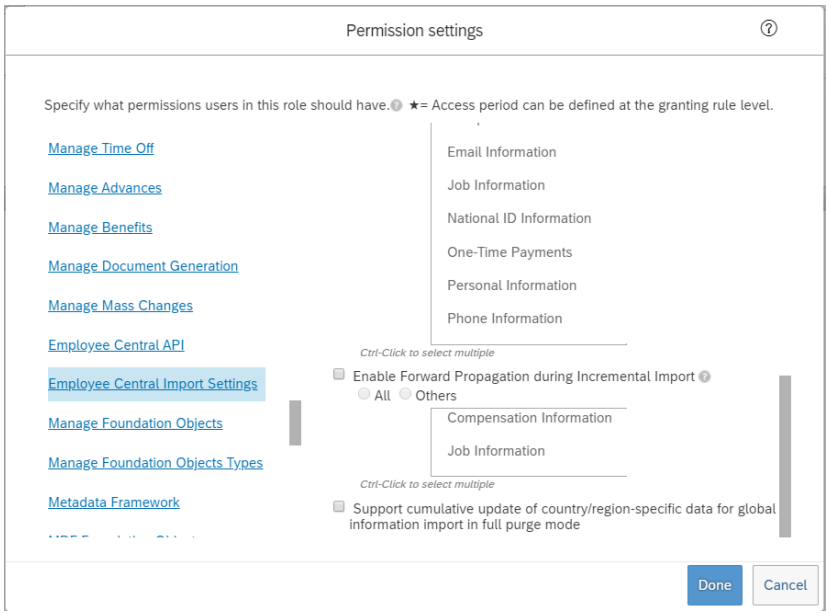
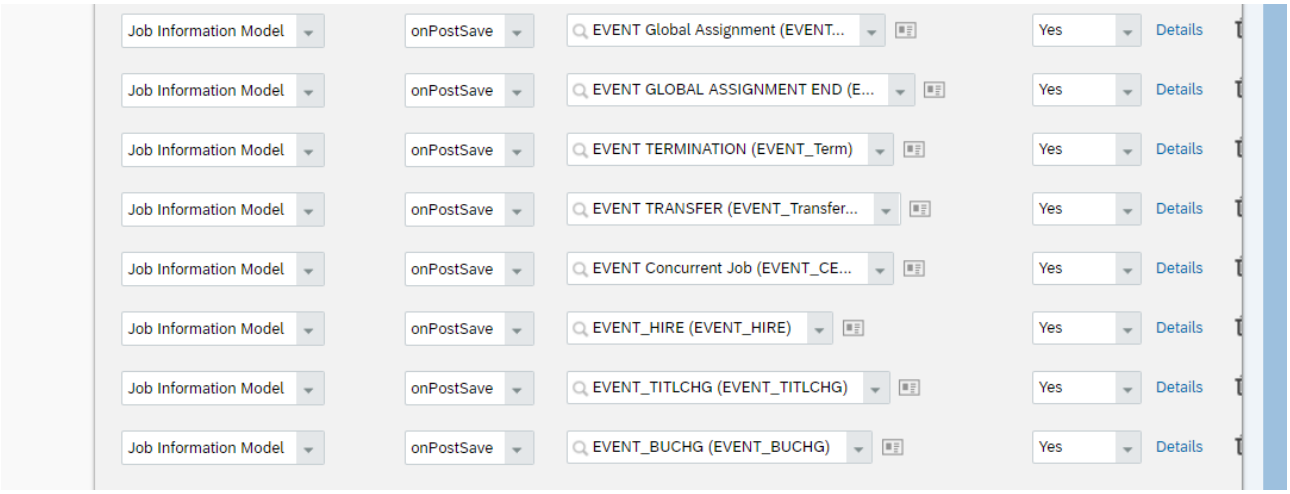


Figure 7: Enabling forward propagation for effective date entities in import and OData

6.3.4 Triggering ISE Events and External Event Notifications

Similar like workflows, ISE Events are triggered by business rules attached to the corresponding object. As a matter of fact, an API call writing data into EC can only trigger ISE Events if the corresponding entity can trigger rules. See rule chapter above.

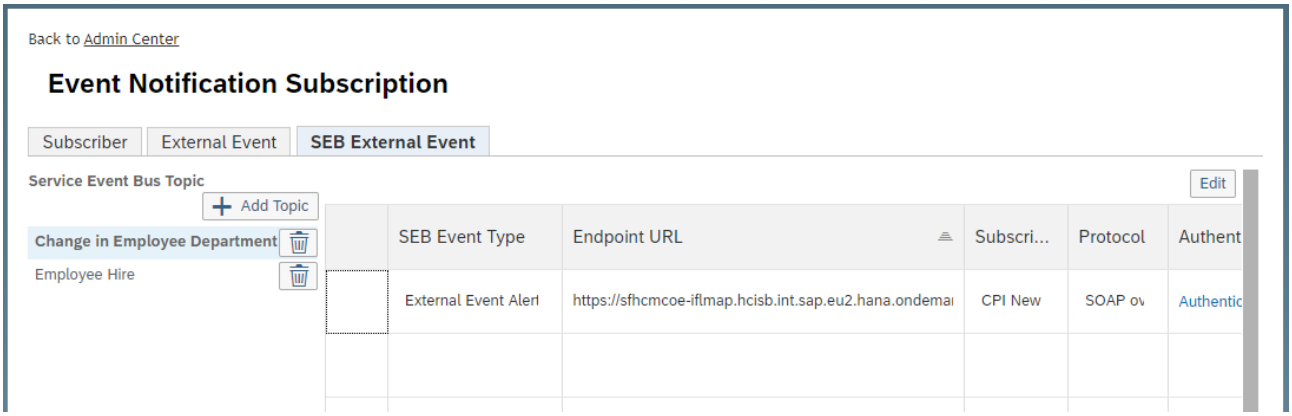
It is also important to mention that those rules have to be triggered at post_save event. Otherwise business rules and ISE Events would get triggered before a workflow approval.



The following blog and KBA explains how to configure the integration center that ISE Events would trigger an integration:
https://blogs.sap.com/2018/04/23/successsfactors-intelligent-services-center-isc-integration-center-ic-replacement-of-3rd-party-middleware-to-trigger-simple-interfaces-from-employee-central/
https://apps.support.sap.com/sap/support/knowledge/public/en/2209164

The interesting part in this internal event driven integration is the capability to define again a special rule which is evaluated before the integration is called. By this the generic event can get triggered but there is the possibility to have an extra condition to trigger the integration.

Another approach of an event driven integration is the usage of external event notifications. In such a case the corresponding event will not trigger the Integration Center but an external listener subscribed via “Event Notification Subscription” from “Admin Center”. Similar constrains as above, the entity in question has to be enabled to trigger rules in case of API calls.



6.3.5 Triggering Workflows

In Employee Central workflows are triggered by special rules during on_save. For MDF objects this can be done in a similar way using rules in the generic object definition or just using a simple default assignment. See also this blog:
https://blogs.sap.com/2016/12/26/triggering-workflows-for-mdf-using-odata-api/

In order to enable triggering of workflows a few pre-requisites have to be met:

- For MDF the setting for OData Admin Access has to be disabled (https://apps.support.sap.com/sap/support/knowledge/public/en/2396714) and for custom MDF objects the parameter worklfowConfirmed=true has to be used in the upsert statement: https://help.sap.com/viewer/d599f15995d348a1b45ba5603e2aba9b/latest/en-US/886cdf72d3474996889d0b306d30c27c.html

- For Employee Central objects the corresponding feature must be enabled by selecting the corresponding entities in the permissions settings of the api user. See figure below.

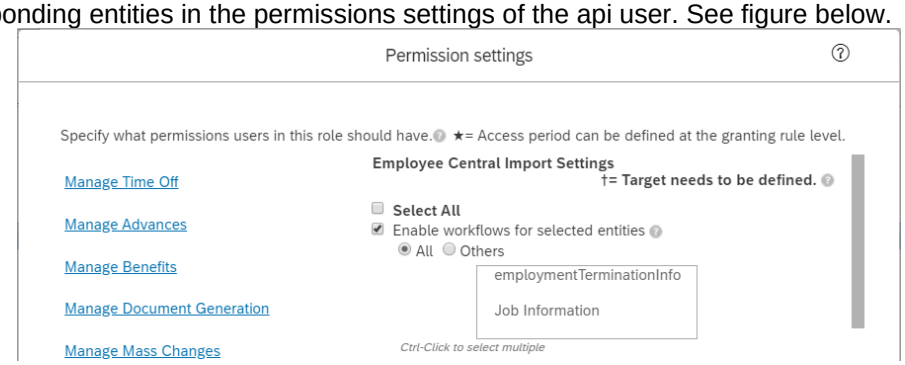


Figure 8: Permission setting in Employee Central to enable triggering of Workflows for API calls and imports

6.3.6 Update changes only

Sometimes data is being replicated into EC from a master system which does not know if there had been any changes to the data. In order to avoid that such unchanged data is being written into Employee Central and changes the audit records which in turn might generate additional delta replications to other system and potentially also events, we introduced a feature in import and the OData upsert to control the behavior.

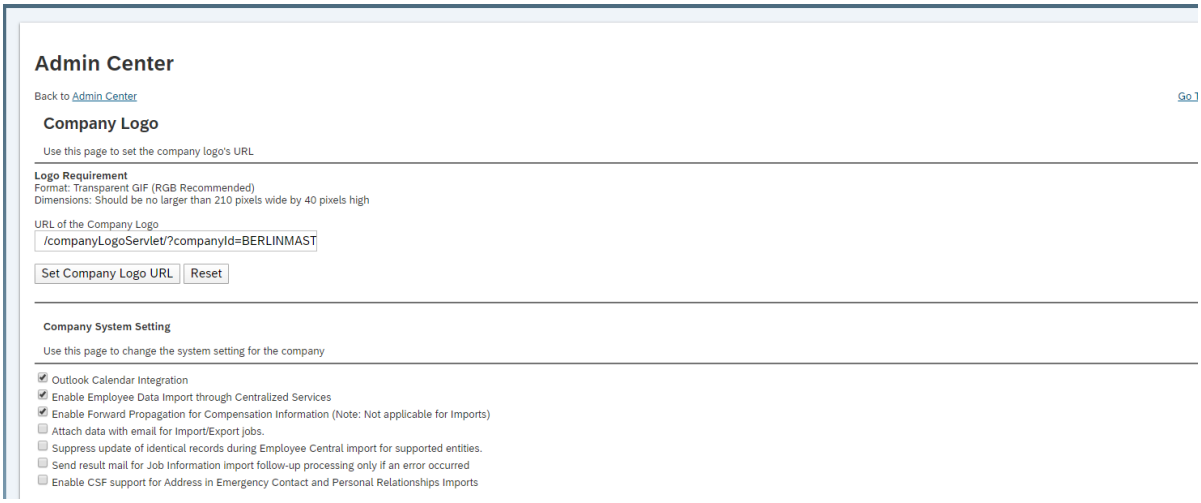


Figure 9: Suppress update of identical records as company setting for imports

In case this feature is enabled in the import UI or used as an URL parameter for UPSERT in OData only changes are being processed and audit entries are not created if there are no changes. See imports documentation and API documentation for details:

- Import - Suppressing Identical Records While Importing Employee Data: https://help.sap.com/viewer/f5c753ba58814ef0ab181747824c41ed/latest/en-US/bc57c11328ad414095d46e648ea142dd.html
- API - suppressUpdateOfIdenticalData: https://help.sap.com/viewer/b2b06831c2cb4d5facd1dfde49a7aab5/latest/en-US/4c65a71d10044f9288060a94f67f4140.html

This is an API example in OData to make use of this feature (without payload and headers):

POST: /odata/v2/upsert?purgeType=full&suppressUpdateOfIdenticalData=true

6.3.7 Full Purge and Incremental

In general, there are two major modes for the upsert OData API and the imports. Full-purge and incremental. They are documented quite well in the corresponding Employee Central API Reference Guide and the Import Guide.

While full purge is a good option for an initial migration/data load it should be avoided in master data integrations. Major reason being that it is changing much more data than required and also creating unnecessary audit records by deleting and recreating the data. In case of an inbound integration writing data into Employee Central we should aim to update records if possible, using incremental mode.

In case of process integrations where always new records are being created and no updates are required, full purge mode and incremental mode have the same behavior and it doesn't really matter which one to use.

Last but not least for User Integration upsert as well as PUT and POST on entity level have limited usage. While MDF and hence custom objects support full permission checks for read and write, the available permission checks for writing data into Employee Central in the context are limited. Only the below mentioned entities and only for file based import support checks on target population if enabled but not field level authorization checks. A new set of Employee Central APIs for Person and Employment objects will address those gaps in future including better roundtrip handling and aligned data models.

6.3.8 Do's and Don't while writing data

In general, we do not recommend to make use of nested upsert statement. In case upsert is used, we recommend to do this in a non-nested approach to gain maximum performance. In case transactional behavior is wanted please use $batch as explained in chapter 5.3.1.

In case of massive write operation into Employee Central we also do not recommend to use PUT or POST based on the entity. Try to use upsert instead. PUT and POST can be used in UI context but do come with some limitations from a feature point of view for integrations.

In several performance measurements we saw that the sweet spot for multiple records in upsert is around 200 records in a single upsert. In case of effective dated objects (also other multi key objects) each effective dated record counts, i.e. if employees have in average 5 effective dated records put 40 employees into one OData request.

Do not do single employee updates using $BATCH or upsert in case you want to update multiple employees this causes issues and in a worst case issues with HRIS Sync.

Parallelization of OData upsert statements might improve the throughput. This is usually not the case for read operations. Parallelization does not have a significant impact here.

Do not do parallel writes using multithreading on the same data. In case you use multithreading limit this to a maximum at 10 threads at any time.

During migration it might be better to do multiple full purge upload rather than updating data after an initial upload since full purge is in general faster than incremental updates. At the same time we recommend to switch of any additional permission checks on target population and also disable rules.

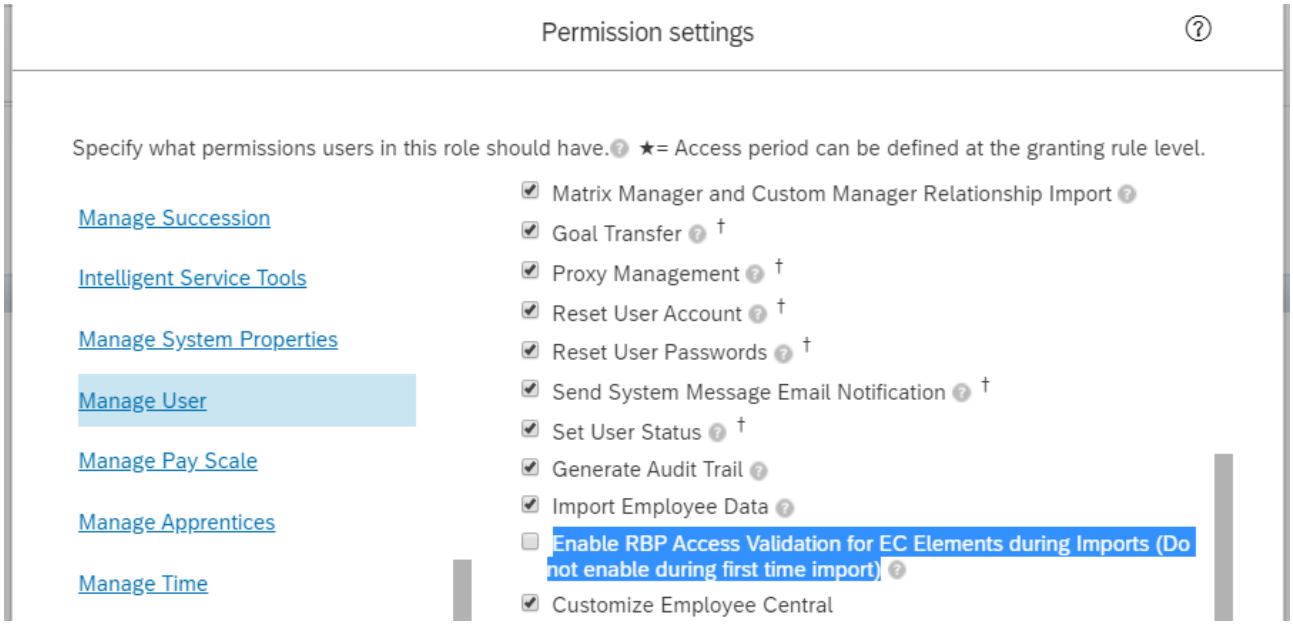


Figure 10: Enabling permission checks on target population for imports

In general user level permission checks for target population or even on field level do not make that much sense when it comes to integration styles such as data integration. In some process integrations it might make sense to have this check, not only for security reasons but also to ensure that audit log entries are not created with an anonymous API user.

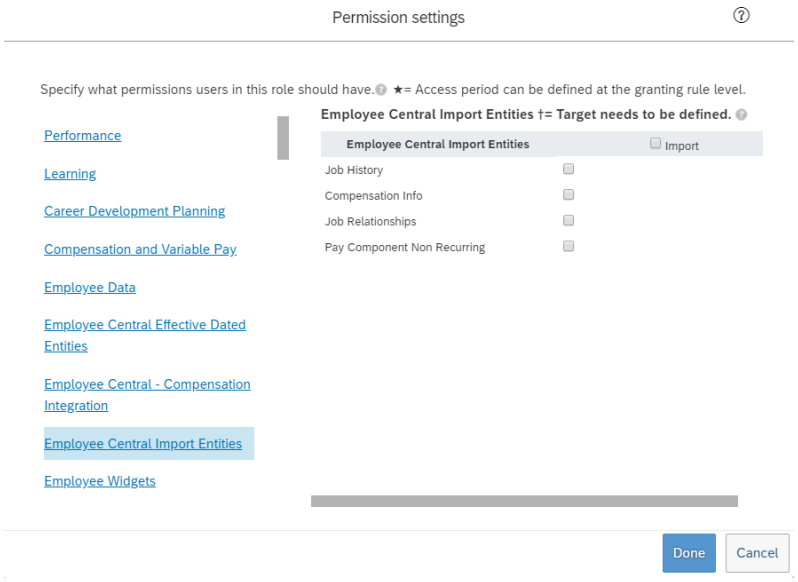


Figure 11: Individual enablement of target population permission checks import file

Those permission checks are available for all MDF objects and the corresponding OData APIs, as well in a limited form also for the above-mentioned file based imports. The EC OData APIs for Person and Employment Entities do not support field level authorization or target population. Hence those shall be used for data integrations styles only or process integration styles where permission and named users are not needed. If needed an own security layer has to be implemented between the corresponding OData APIs and the application calling those. This can be achieved by a combination of IP filtering and an active part between the application and the SuccessFactors backend.

Don't do Basic User Imports or OData User Entity writes for data which is coming from Employee Central structures. This data will get overwritten by HRIS-Sync, always save into EC Entities in case of writing into EC. Only basic user information which is not coming from Employee Central shall be written, e.g. username in case of integrations to other identity providers, other Identity relevant information such as email and telephone

number have to be written into the EC person entities though. In such a case ensure that no HRIS Sync Job is running at the same time.

With only a few exceptions all integrations should not have detailed authorization checks on field or record level. Technical users shall be setup in a way that they see all the required data using the admin API permission setting explained in the corresponding guides. Only exception for this are user integration styles and in cases where company regulations or laws enforce the customer to apply permission checks also for technical integrations, this can be the case for subsidiaries within a global ultimate sharing a SuccessFactors tenant but also in case of not trusted 3rd party integrations. In all other cases SAP does not recommend doing those permission checks since filtering data by permissions will cost additional time and slow down the integrations, it also comes with a certain risk that due to missing authorization data is only partially written or read which might cause integrations to fail. Such side effects caused by permission checks are very difficult to track.

6.3.9 Summary of Side-effects

There are several side effects caused when data is written into Employee Central. A side effect in this case is the creation or change of data caused by the initial attempt of writing data. These side effects might cause confusion in case data looks different than expected after writing. Those side effects include:

- Business Rules which can change or create further data
- Forward propagation in effective dated objects
- Intelligent Services triggering Actions or Integration in SAP Integration Suite or Integration Center
- HRIS-Sync changing the user entity after Employee Central objects are changed (might be even delayed in case of asynchronous execution), see this KBA for details of hard coded mappings
- Additional Business Logic such as Position Management, Global Assignment or Employee Time where jobs are triggered to adjust data
- Additional record level permissions might cause data not being written as defined in the corresponding API

6.4 Managing Dates, Times, Time zones and Daylight saving

The most common use case to use audit information, such as last modified queries in OData, are data integration styles, in specific the master data synchronization use case pattern. The general recommendation in any master data synchronization which transfers a significant amount of data (rule of thumb should be everything's which reads above 1000 records a day) is to use a delta approach and transfer only changed and new created data. This record or object level delta loads are based on a similar pattern using last modified queries across required entities. The following figure illustrates this.

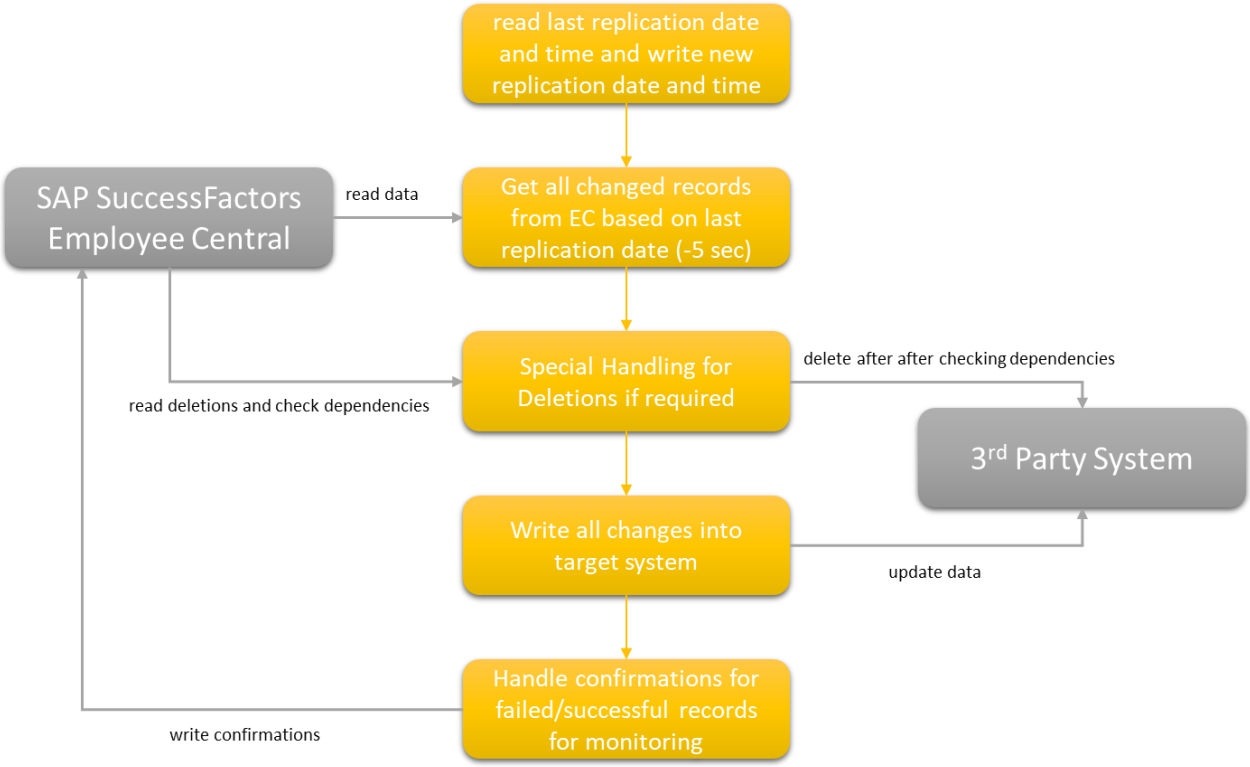


Figure 12: Typical flow of a data integration to replicate master data with a record level extraction pattern

In general, the flow is as follows, based on the date and time of the last replication data is being read from EC using last modified queries to get changes only. In some case a special logic might be require covering deletions, sometimes this is already included in the reading part. See also chapter 5.5 for this. After those dependencies are checked data can be deleted and updated and the last replication date adjusted. Confirmation messages can be created to ensure that failed synchronizations are fixed and trigger another replication (another approach is the usage of a staging area in the 3rd party system).

In order to ensure that records are not lost or are retrieved several times, those are the recommendations with respect to date and time handling when building integrations using a delta extraction pattern on object or record level in OData:

- In case OData is being used move the last modified date at least 5 seconds before the last replication date to ensure that delayed updates or small differences in timestamps are not causing missing records. Compound Employee API does this automatically.
- Use fromDate and toDate to get history information or asOfDate to get a specific record in time, the default behavior is asOfDate with the current date
- Use the lastModifiedDateTime fields instead of the lastModifiedOn fields to ensure that you get all of the information in UTC and not in a local server timezone, see details here (SAP, 2019)
- Store your replication start date and time in UTC too to avoid conversions, use the execution timestamp provided by the Compound Employee API query or the APIQueryExecutionTimestamp in the headers of the response in OData for this.
- Always put the last modified date filter on the left, e.g. $filter=lastModifiedDateTime+gt+… otherwise deletions will not get detected (only OData)
- Consider that last modified queries together with fromDate and toDate have a special behavior, a last modified query return all effective dated records of the changed person or employment in case one of the records have been changed (entity level delta instead of record level delta) see also (Help SF OData LM)
- That it is more convenient to work with $format=ATOM to see human readable UTC times instead of EPOC timestamps in JSON
- Not that the time zone of the API server is returned as part of the OData API header in the response in parameter SFODataServerTimeZone
- Consider the different behavior between datetime and datetimeofffset field. Use datetimeoffset with lastModifiedDateTime querries and datetime with lastModifiedOn querries.

6.5 Performing and Detecting deletions

6.5.1 When and how to perform deletions

SAP SuccessFactors offers several capabilities to perform hard deletes in the system. The table below shows the set of available options for different entities and protocols including relevant documentation. Note that due to the side effects of hard deletes those should be avoided in general.

Option	Supported Entities in EC	Supported Technology	Comments
Entity CompoundDelete (see this KBA)	Job Relationships, Pay Component Non-Recurring, Addresses, Email Information, Phone Information, National ID Information, Biographical Information	CSV, (OData)	OData Entity is restricted
Delete/Delimit toke in operation fields of Entity (Import Help)	Addresses, Phone Information, Email Information, Social Accounts Information, National ID Information, Emergency Contact, Job Relationships, Pay Component Recurring, Pay Component Non Recurring, Direct Deposit, Person Relationships, all MDF	CSV, OData (upsert only)	
Delete http operation (HXM Suite OData Developer Guide)	MDF, Background elements	OData	
Implicit delete with full purge (KBA, Import Help, HXM Suite OData Developer Guide)	Biographical Information, Job Information, Addresses, Phone Information, Email Information, Social Accounts Information. National ID Information. Emergency Contact, Job Relationships, Pay Component Recurring, Pay Component Non Recurring, Direct Deposit, Person Relationships, all MDF	CSV, OData	will only work if there is at least one record left

Table 2: Available options to perform hard deletes, SOAP version are not mentioned here since they are all deprecated

From the options above a quite unknown one is the one how to delete an Email entry (Olsson, 2018). This is an example of a payload to delete an email record for an employee with person Id 103223 and email type 8448. We used the API URL for our salesdemo environment here:

POST https://apisalesdemo4.successfactors.com/odata/v2/upsert

Authorization : Basic <your base64 encode of "Basic: apiuser@company:password">

Content-Type : application/json

{ "__metadata": { "uri": "https://apisalesdemo4.successfactors.com/odata/v2/PerEmail(emailType='8448',personIdExternal='103223')", "type": "SFOData.PerEmail" }, "operation": "DELIMIT" }

Another option in OData is the upsert function import. It can be used to replace existing records with new ones and hence also deleting existing ones. This implicit deletion works fine as long as there are records left. If the records of an entity shall be deleted completely, this approach will fail. The only solution to do a hard delete for the last remaining record is the delete/delimit option mentioned above.

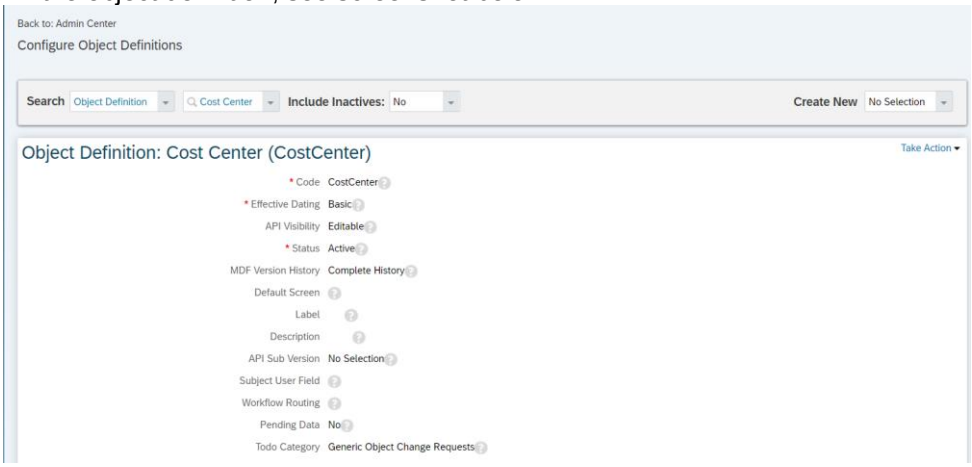
As mentioned, it is our recommendation to avoid hard deletes and use soft deletes instead which is possible for foundation objects for example. The following list is a collection of use case where a hard delete might make sense:

- Wrong data was written into the system by accident and usage by other entities can be ruled out
- Data must be deleted for data privacy or compliance reasons and dependencies will be removed before
- Soft deletes are not possible without confusing users or overloading UIs

In cases where new entities are being created using MDF and where an integration to other systems are planned, someone should avoid hard deletes and introduce a soft delete concept. If soft deletes are used there is no special handling required for integrations.

6.5.2 Pre-requisites and limitation for tracking of hard deletes

A pre-requisite for deletion tracking is the enablement of change audits (at least delete audits), as long as no audit records are written in the system the APIs can also not retrieve those. For MDF objects this setting has to be enabled in the object definition, see screenshot below:



The MDF guide (SAP, 2019) explains the difference between the MDF Version History audit options. To detect deletions the setting “deletion history” is enough for this, someone does not need the complete history.
For EC person and employment objects those audit logs are always created, hence there is no need to enable them.
For OData it is defined that the audit records are only included in the query, if the last modified parameter is on the left side of the $filter condition. If it is on the right side the audit entries are not included in the query. Only if audit records are included in the query all changes including deletions can be tracked. This is true for EC entities such as Person and Employment but also MDF entities. See also here:

- Employee Central Entities and APIs: https://help.sap.com/viewer/b2b06831c2cb4d5facd1dfde49a7aab5/latest/en-US/73cfd57b6db440adb7b8a6f5d5ac018a.html?q=right%20last%20modified
- MDF Entities and APIs: https://help.sap.com/viewer/e4a4ce68589841709a8202928c23803a/latest/en-US/4af1754504b04e0d9ccc773bf0b26334.html?q=last%20Modified

It is also important to know that those audits will be limited in future for a maximum of 3 months, i.e. queries with lastModifedDate more than 3 months in the past will not be supported anymore and usually make also no sense. This does not mean that history information which is older than 3 months cannot be read it just means that in case of a delta query used in a master data replication the longest supported schedule in which data can be synched is 3 months. If a target system has not been updated with current data for more than 3 months a full load will be required to sync the systems again.

Another important setting is the ability to track deletions in OData for expanded entities. This allows tracking of complete deletions as long as the root entity is still available, see also the "SAP SuccessFactors Employee Central OData API: Reference Guide": https://help.sap.com/viewer/b2b06831c2cb4d5facd1dfde49a7aab5/latest/en-US/dbdb73d752184792a1addaf60c72f0ce.html?q=Deleted and screenshot below.

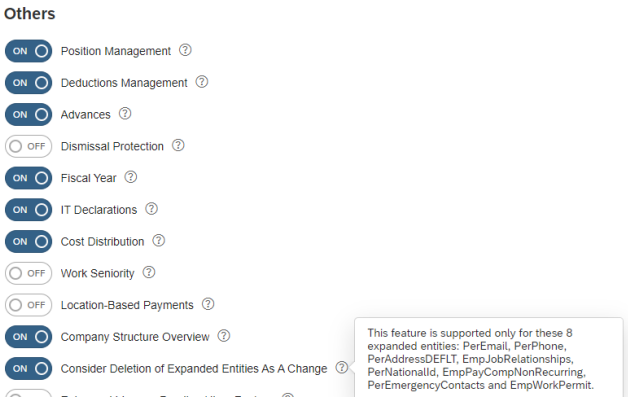


Figure 13: Mandatroy setting to enable tracking hard deletes using OData for selected entities as part of "Manage Employee Central Settings" in Admin Center

6.5.3 Options for tracking of hard deletes

The Compound Employee API can cover deletion detections for all supported entities. It is the first and best option to be used when it comes to master data synchronization of the supported entities. The major downside of this API is the missing support for foundation object changes. It supports several modes of change tracking including field level delta. Detailed documentation for the employee/object level delta and field level delta modes can be found here:

Implementing the Employee Central Compound Employee API

A 2nd option to track hard deletes is a last modified query on an entity. The major downside of this approach is the missing support for a full deletion of the records. E.g. while a last modified query on JobInformation will still return all remaining records as long as there is a record left for an employee, this is not the case anymore if the last record is being deleted. While this is usually uncommon for Job Information record it might be quite common for Job Relationships which have the same limitations. While effective dated objects support those partial deletes, non-effective dated records do not support those at all. Hence the only option to keep another system in sync in this case is a business key comparison explained in the next chapter.

A more practical option to cover partial and full deletes in OData is the usage of an indirect query using a different root entity. A query like this:

GETodata/v2/PerPerson?$filter=emailNav/lastModifiedDateTime+gt+datetimeoffset'2020-01-20T15:55:23Z'&$expand=emailNav

Will return all email records of a person no matter if the email was created, changed or deleted. Even if all Email have been deleted the PerPerson entity will be returned with an empty email navigation.

We do recommend to use this approach, join several OData entities to create a delta query only as long as the complexity of the query keeps in certain boundaries. If this is not the case we always recommend to use the Compound Employee API where suitable or use a process as described in the next chapter.

Option	Supported Entities in EC	Supported Technology	Comments
Compound Employee API	Person, personal_information, address_information, phone_information, email_information, person_relation, employment_information, job_information, compensation_information, paycompensation_recurring, paycompensation_non_recurring, payment_information, accompanying_dependent, alternative_cost_distribution, job_relation, direct_deposit, national_id_card, deduction_recurring, deduction_non_recurring, global_assignment_information, ItDeclaration, dependent_information, personal_documents_information, EmployeeDataReplicationElement, associated_employee_information, emergency_contact_primary, DRTMPurgeStatusOverview	SOAP	Supports changes and deletions including full deletions of sub-structures (e.g. delete all emails of an employee). Does not support foundation objects or non-user related MDF entities.
Direct query of Effective Dated OData Entities in Employee Central	EmpJob, EmpCompensation,...	OData	Supports changes and deletions, as long as there are records left.
Direct query of non-effective Dated OData Entities in Employee Central	PerEmail, PerPhone, Employment, …	OData	Supports changes only but no deletions, even not partial ones
Indirect query of Employee Central Entities with "Consider Deletion of Expanded Entities as a Change" enabled	PerEmail, PerPhone, PerAddressDEFLT, EmpJobRelationships, PerNationalId, EmpPayCompNonRecurring, PerEmergencyContacts and EmpWorkPermit.	OData	In this case the OData query returns the root entity records only in case all records of the related entity are deleted
MDF OData Entities		OData	

Table 3: Available option to detect hard deletes

6.5.4 Handling hard deletes

There are some cases where hard deletes, if not avoided, can't be detected using a single API call. This is for example the case for completely deleted MDF objects, completely deleted foundation objects (e.g. Cost Center). It is important to note that it might not be a good idea to replicate deletions of those objects into other systems since the number of dependencies in those systems might be different than in the source system and deletion might cause issues. SAP recommends to create a special 2 step approach for such integrations or create independent reports in those system to check for dependencies and delete entities only if those dependencies are resolved.
In case the integration and not a tool in the target system shall take care about deletions this is a viable approach:

1. In the first phase replicate all changes of the foundation or MDF objects using the OData API and a normal lastModified query
2. In the 2nd phase manage the deleted objects (all completely deleted objects)
    a. compare the keys of all available records in the target system with the keys of all available records in the source system to define the deleted ones. In case the retrieval of keys is not possible from the target system store the replicated key in a 2nd persistency which is managed in the integration process.
    b. Before deleting the records in the target system ensure that there are no dependencies created which would create issues such as data inconsistencies after a deletion. If this can be excluded, perform a hard or a soft delete.

6.6 Accessing pending Data in EC Workflows with APIs

As of today there is no event to trigger an external system in case a workflow is created. The only option to become aware of a created workflows is to poll for created pending data or created workflow request. In this chapter we like to highlight the solution to build two scenarios of HR workflow reporting and the ticketing system integration.

For the use case of HR reporting it is important to mention that the recommended solution is always to use the products built in reporting capabilities instead of building a custom solution. In case this is not enough the following chapter explains how to access workflow data including pending data for EC entities including MDF.

Before going into the details of the solution we like to introduce the simplified data model for the OData Entities of the workflow. Details can be found in the SAP SuccessFactors Employee Central OData API Reference Guide in chapter Workflow. (SAP, 2019)

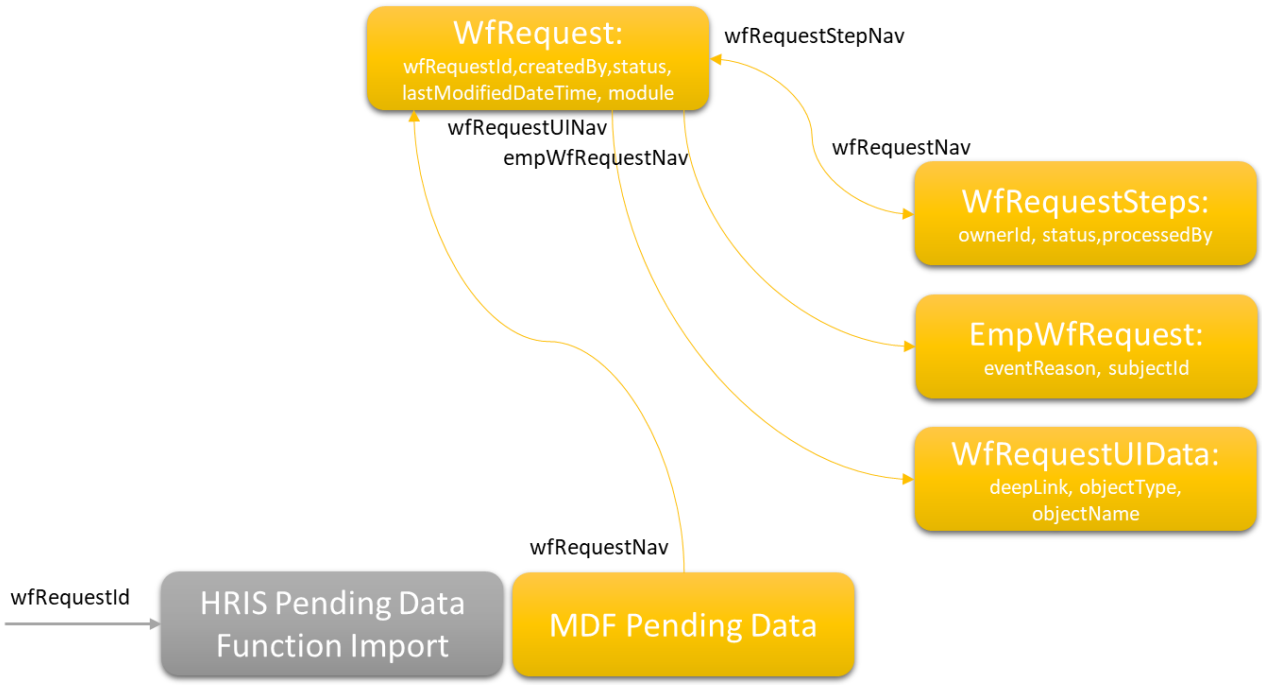


Figure 14: Simplified Workflow OData Entity Metadata

This data model starts from the WfRequest entity. The major information from this entity is the unique ID of the workflow called wfRequestId, the creator of the workflow (createdBy) as well as the overall status of the workflow and the last time the workflow was changed (lastModifiedDateTime). The module field contains either HRIS in case we have a workflow for EC Entities or GENERIC_OBJECT in case this is an MDF Workflow. With this information we can already retrieve all open workflows which have been created by a specific user or are in approval with a user.

If we need to get all open workflows we just use a filter on the status ($filter=status+eq+'PENDING') and if we need all open workflows created by users in a specific department we have to retrieve those user IDs with another OData call first and use an IN clause in WfRequest to retrieve all such workflow requests:

$filter=status+eq+'PENDING'+AND+createdBy+IN+'user1','user2','user3'

You might have to use $batch in case you hit the URL length limitations or split up your request into several ones. In case someone wants to see all open workflows which are waiting for approval for a set of responsible users in your department this is not that easy anymore since the workflow request does also allow for dynamic groups where there is no unique user responsible. In other cases where there is an owner ID in WfRequestStep, we can use a deep filter on WfRequest:

$filter= status+eq+'PENDING'+AND+wfRequestStepNav/ownerId+IN+'user1','user2','user3'

There is another case of users associated with workflows. So far we talked about the creator of the workflow,i.e. the person triggering the workflow request being stored in createdBy of Entity WfRequest and the owner to approve this workflow request, which is stored in WfRequestStep in field ownerId (in case there are multiple owners we might find the position, dynamic rule or group here). The third person is the person whose data is being approved in the workflow. This information is available only in Entity EmpWfRequest in case the corresponding workflow module is HRIS. In this case the user ID of this person (please note that a person might have several userIds in case of global assignment or concurrent employment) and the event reason for the change can be found in fields subjectID and eventReason. Again, a deep filter on WfRequest can be used to retrieve all workflows for a set of userIds using a IN clause for the field subjectId.

Finally, there is also an Entity called WfRequstUIData which offers additional information about the UI deep link to the workflow, which allows someone to call the workflow UI to see all the data and take action. So far there had been no big differences between MDF and EC entities. Both are using the same workflow entities. For pending data this is different in many ways. MDF uses the same entity as the core entity (e.g. Position or EmployeeTime) a proprietary URL parameter called record status to store pending records while EC used one function import with a single workflow ID parameter to access this data.

As a result of this architecture MDF the first 10 pending workflows for Positions created by user sfadmin can be retrieved in one shot by using a deep filter:

GET odata/v2/Position?
recordStatus=pending&
$expand=wfRequestNav&

$filter=wfRequestNav/status+eq+'PENDING'+and+createdBy+eq+'sfadmin'&
$top=10

This kind of navigation works only for pending data, in case someone wants to see pending history or already approved workflow requests the deep filter will not work anymore. In such case the only option to get the approved data is to use the information from WfRequestUIData (objectName and objectType).

For EC workflows this has to be a two-step approach. The first step is retrieving the workflow information, in this example the first 30 pending workflows created by user sfadmin:

GET odata/v2/WfRequest?
$filter=createdBy+eq+'sfadmin'+and+status+eq+'PENDING'&
$expand=empWfRequestNav,wfRequestUINav,wfRequestStepNav&
$format=JSON&
$top=30

This information can be used, to be precise the workflow IDs, to feed them into the pending data function import from Figure 14 above. Since this function import only supports single parameters for workflow ID, we should use $batch in order to avoid too many single API calls. See also chapter 5.2 for batching using $batch. An example call for getting pending data for EC workflows looks like this, were 3705L is the corresponding workflow ID coming from the request above:

POST odata/v2/getWorkflowPendingData?
wfRequestId=3705L&
$format=JSON&
$expand=workflowAttributeGroups

The same 2 step approach is also possible for MDF workflows, this is recommended in case the pending data is only needed on request. In such a case the 2nd query would be again a deep filter for the MDF objects using the workflow request IDs and an IN clause:

GET odata/v2/Position?
recordStatus=pending&
$expand=wfRequestNav&
$filter=wfRequestNav/wfRequestId+in+'5761','5762'
The example above is a simple request to retrieve two workflows with IDs 5761 and 5762.

With the information above we have all we need to build our HR pending workflow use case. We can retrieve different types of workflows using a WfRequest query, we can enrich it with pending data in a 2nd step or for MDF also in one shot. In case we need allowed actions to be performed on those workflows there is a 2nd class of workflow entities that becomes relevant. Those are the 5 function imports to approve, reject, comment, withdraw or sendback a workflow. Before someone can call those function imports it has to be ensured that the caller of the API is also the owner of the corresponding step and that the operation can be executed in general. This information can be retrieved from the entity WfRequest by expanding the navigation to workflowAllowedActionListNav. Examples are available in the SAP SuccessFactors Employee Central OData API Reference Guide in chapter Workflow. (SAP, 2019)

When it comes to triggering a third party system when a workflow is being created, e.g. a ticketing system, it is usually required to

- create new ticket for new workflows
- update tickets for changed workflows (including withdraw, approval and cancellation)
- in some cases, approving a workflow step based on the status of the ticket in the ticket system

The difference to the use case above is that we now need to keep track of changes on workflows. E.g. each time a workflow is being changed or created we have to trigger the creation or update of a ticket. As a result, the first query will have to be a last modified query where the date of the last synchronization with the ticketing system has to be stored somewhere, e.g. in the middleware or in any of the two backends. Depending on the date and processing of the last synchronization the query to retrieve this delta information of new created or changed workflows will look like this:

GET: odata/v2/WfRequest?
$filter=lastModifiedOn+gt+datetime'2019-11-01T12:00:00'

In case only newly created workflows are of interest a query on createdOn can be performed. Based on this data and maybe additional expanded information using additional $expand URL parameter a ticket can be created in the ticketing system. The same rules apply again as in the use case before to retrieve pending data. The major difference will be how to handle changes on existing tickets based on changes in the workflows. This will require analyzing the data of the workflow, e.g. if the status of a workflow changes and adjust data in the ticketing system accordingly. So far the process integration went just into one direction, creating tickets and updating them based on changes of the workflow. In case the process integration shall also go into the other directions we have to call again the workflow function imports to take action on the workflow. In this case we assume that the workflow configuration was setup in a way that a workflow step has been created to allow the ticket system to approve this step as soon as the ticket is completed and allow the workflow to continue. To allow this a few things have to be ensured:

- As mentioned before a workflow step has to be part of the workflow configuration
- The ticketing system has to store the reference to the workflow or employee central has to store this information in a custom MDF object
- The user responsible to approve the workflow step and the user used from the ticketing system to call the function import for approval have to be the same

With respect to the last pre-requisite the recommendation would be to use technical user only in case the users using the ticketing system are not known to EmployeeCentral and it is not important to have an audit log who closed the step by closing the ticket. Approving the workflow step from the ticketing system can be done in a scheduled way using a middleware by retrieving closed tickets and approving workflows accordingly or in a direct if the ticket application can be extended by calling the API directly as soon as the ticket is being closed. At this step the information about the link between the ticket becomes important again and has to be used to map the ticket number to a workflow number before calling the workflow approval API.
In case there must be a link visible between the workflow and the ticket from employee central for other approver of the workflow one option is to use the comments in the approval to create an information about the ticket. This comment can be passed from the ticketing system when the corresponding workflow request step is being approved.

7. ASSUMPTIONS AND EXCLUSIONS

Through the whole document we assumed that the API usage is for either data integration or process integration. We excluded in this document any recommendations for user integration or thing integration.

Most recommendations in this document should be independent of the used integration platform.

8. REFERENCES

SAP Notes/KBAs

- 2613670 - What are the available APIs for SuccessFactors?
- 2735876 - Odata API Best Practices [Query Modified Records, Pagination - Batch Size, Timeouts, etc.,] - SAP Successfactors Odata API
- Representational state transfer

SAP Blogs

- Integration Solution Advisory Methodology (ISA-M): Define Integration Guidelines for Your Organization
- CPILint version 1.0.0 is ready

SAP Help Portal

- SAP SuccessFactors Employee Central OData API: Reference Guide - Workflow
- Overview (OData Version 2.0)

- SAP SuccessFactors HXM Suite OData API: Developer Guide – About HXM Suite OData APIs
- SAP SuccessFactors Employee Central OData API: Reference Guide – Getting your time zones right
- SAP SuccessFactors Employee Central OData API: Reference Guide – Last Modified Date Time and $filter
- SAP SuccessFactors HXM Suite OData API: Developer Guide – Querying Effective – Dated Entities

Adiitional Resoucre

- API and Integration
- Retrieved from Delete/Remove email address in PerEmail entitiy with SAPSF
- Implementing The Metadata Framework

9. APPENDIX

9.1 API Restrictions

Table 4 below lists restrictions which must be considered when calling REST based APIs in general or SuccessFactors APIs in specific. The scenario describes the aspect of the API which is effected by the restriction while the restriction column defines the limit. In case the restriction is valid only for a specific Technology (OData, SOAP), Operation (POST, GET, upsert) or Entity (e.g. PerPerson) the corresponding three columns indicate this.

Scenario	API	Operation	Entities	Restriction	Details/Solutions
URL Length	OData	All	All	2k, 2047 characters for some clients, up to 8k for API servers	KBA; use $batch or simplify request
Maximum payload size	OData	POST	All	200MB per request	Split requests
Maximum records in upsert	OData	POST, upsert	All	not more than 1000 records per request	Help.sap.com chapter 5 attachments; split requests
Backround Elements do not support full purge	OData, SOAP	POST, upsert	All Backround Elements	No support for full purge	use delete instead
Function Imports for Permissions	OData	POST	getRolesPermissions, getUsersPermissions	Maximum 100 userIds per request	Split request accordingly
Attachment Size Limit	OData	POST	All Entities supporting Attachments	5 MB per Attachment	Help.sap.com chapter 5 attachments
Maximum number of Expanded Records	OData	GET, $expand	All	300000 expanded records	KBA; split requests
Maximum number of values for IN clause	All	GET, $filter	All	Maximum of 1000	Help.sap.com; split requests
Deduction Entities not write enabled	OData	POST, PUT, upsert	RecurringDeduction, OneTimeDeduction, RecurringDeductionItem	Does not support write via OData	KBA; use imports
$batch maximum number of request	OData	POST	All	Maximum number of request in each $batch request is 180	KBA; split requests
OData Timeout	All	All	All	10 minutes	KBA; reduce requests or retry
Time in months Audit	All	GET, query	All	Audit records are kept for a maximum	KBA

records are kept				amount of 3 months in future.	
Maximum number of records per http session for CE API	SOAP		CompoundEmployee	Maximum of 1 million records per http session	Help.sap.com chapter 3.6 server paging; split requests
Service Limits for Snapshot-Based Pagination requesting always the same first page	OData	GET	All	30 on company user level and 5 on entity level	Help.sap.com chapter snapshot based pagination; correct integrations; use different paging during testing
Function Imports	OData	POST, GET	e.g. getUserRolesByUserId, getWorkflowPendingData, …	Not supported in Integration Center	Use SAP Integration Suite instead
Parallel Threads for read or write	OData	POST, GET	all	Maximum of 10 parallel threads at any time	Schedule integrations at different times, reduce API complexity and response size and use delta if possible

Table 4: API restrictions and their solutions

9.2 Automatic and manual checks for good integrations

This lists some check to analyzes the quality of integrations. It explains the steps to be executed manually for the check and the criticality or impact of an integration with such a flaw. Now those checks are based on the API Log but can be performed in many cases also on the integration content if this content is accessible.

Automation of those checks might be delivered in the future with SAP SuccessFactors check tool but can also be built by partners till then using tools such as SAP Business Technology Platform or SAP Integration Suite.

For checking SAP Integration Suite content against development guidelines, Morten Wittrock provided an open source tool in this blog (Wittrock, 2019).

Check	Steps	Automation	Criticality / Impact
Data Integrations should not use permissions checks: Check the API logs for calls with high payload and check the permissions settings for the used API user	1. Check the API users being used in the OData API Log 2. Check the permission settings for this user in view user permissions	Use restricted OData Entities ODataApiAuditDetail and ODataApiAudit for OData API log and the function import getUsersPermissions to automate.	Performance

Last modified queries longer back than 3 months: Check the last modified date given in API calls if older than 3 months compared to API log entry	1. Export the OData API Audit Log using Rest Client for Entity ODataAPIAudit 2. Search the OData API Audit log for queries using last modified date in $filter which is older than 3 months	Use ODataApiAudit in OData or Splunk to automate.	Correctness, lost records in replication
Check for many single record API calls of same type, e.g. thausends of full key entity access in OData or CE API with one person ID	1. Export the OData API Audit Log using Rest Client for Entity ODataAPIAudit 2. Upload into Excel 3. Sort by time 4. Filter for specific Entity 5. Filter for API users only 6. Check for many similar calls with a single value for filter, e.g. userId+eq+'123'	Use ODataAPIAudit Entity to automate steps 3-6 on the left	Performance
Check for not using session reuse: amount of API calls compared to amount of logins	1. Export the OData API Audit Log Detail using a Rest Client for Entity ODataAPIAuditDetail 2. Upload into Excel 3. Check URL Request Header for usage of Cookie and X-CSRF-Token 4. Almost every call should do session reuse	Use ODataAPIAuditDetail Entity to automate steps 3-6 on the left	Performance; Help.sap.com
Check for API calls reading exactly the same data again (should be cached)	1. Export the OData API Audit Log using Rest Client for Entity ODataAPIAudit 2. Upload into Excel 3. Sort by API user and URL 4. Check if many equal API calls are made with the same URL 5. Typical entities to look out for are starting with FO or Picklist.	Use ODataAPIAudit Entity to automate steps 3-5 on the left	Performance
Check for productive usage of OData APIs in /restricted/odata/v2 and /beta/odata/v2. Those should not be used.			Stability and Performance

Check Entities being used in wrong context, e.g. calculated fields in integrations	1. Export the OData API Audit Log using Rest Client for Entity ODataAPIAudit 2. Look for intense usage of empCompensationGroupSumCalculatedNav in $expand statements of OData	Use ODataAPIAudit Entity to automate the check for this expand	Performance
Bad usage of paging	1. Export the OData API Audit Log using Rest Client for Entity ODataAPIAudit 2. Check that $TOP and $SKIP is not used with API users (not reliable) 3. Check that snapshot based paging is used in URL parameter (link) 4. Check that all calls with snapshot paging are making use of session reuse	Use ODataAPIAudit Entity to automate the check for this expand or check the iFlow for right paramters	Performance and Correctness
Check for Frequent Full Loads	1. Export the OData API Audit Log using Rest Client for Entity ODataAPIAudit 2. Check for repeated OData querries reading all data and not using last modified querries. Look for high amount of returned records		Performance
Unsecure API calls using Basic auth instead oAuth.	1. Basic authentication shall only be used for testing and also in this case only if a communication outside of the SAP network can be ensured. Even in Preview or Test systems if might be a security risk if passwords go through the internet. 2. Check if API calls in API log make use of Basic Authentification, by checking the API Request URL Parameter Authenticiation: If the string contains Bearer it uses oAuth if it contains Basic it uses Basic Authentification	Automate based on ODatAPIAuditDetails	Security

Table 5: Integration Quality Checks with their impact and manual steps to execute.

9.3 Frequently asked questions

Question	Answer	Link
My APIs are suddenly failing even though I didn't change anything	APIs might start failing in case the availability of fields have been changed in the data model. This might happen even delayed since the change in the data model will be reflected in the API only after a job finishes and the refresh of the OData API metadata completes.	https://help.sap.com/viewer/b2b06831c2cb4d5facd1dfde49a7aab5/latest/en-US/4431834f5a404d289a2fabd2fffa8d9f.html
OData API does not return changed or deleted records?	In case of MDF entities check if the history is enabled in the object definition Ensure that the last modified parameter in the $filter condition is on the left side of the condition, e.g. lastModifiedDateTime+gt+datetimeoffset'2010-01-20T14:30:00Z' Ensure that the deletion of the record happened after your filter condition. Be aware that the corresponding timezone of the last modified field might be in UTC or server time zone	
I got this error: <?xml version="1.0" encoding="utf-8"?> <error xmlns="http://schemas.microsoft.com/ado/2007/08/dataservices/metadata"> <code>ServerErrorException</code> <message lang="en-US">Index: 0, Size: 0</message> </error>	This error happens in case the URL contains 2 comma's in a row ",,"	
Where can I find explanations for API error messages returned in the response body?		https://help.sap.com/viewer/d599f15995d348a1b45ba5603e2aba9b/latest/en-US/136931f720dd4b588eccb422927f041.html
Where do I find login errors and explanations?		https://help.sap.com/viewer/d599f15995d348a1b45ba5603e2aba9b/latest/en-US/9bf9045cd75c400395f018d1afa104f3.html

www.sap.com/contactsap

© 2020 SAP SE or an SAP affiliate company. All rights reserved.
No part of this publication may be reproduced or transmitted in any form or for any purpose without the express permission of SAP SE or an SAP affiliate company.

The information contained herein may be changed without prior notice. Some software products marketed by SAP SE and its distributors contain proprietary software components of other software vendors. National product specifications may vary.

These materials are provided by SAP SE or an SAP affiliate company for informational purposes only, without representation or warranty of any kind, and SAP or its affiliated companies shall not be liable for errors or omissions with respect to the materials. The only warranties for SAP or SAP affiliate company products and services are those that are set forth in the express warranty statements accompanying such products and services, if any. Nothing herein should be construed as constituting an additional warranty.

In particular, SAP SE or its affiliated companies have no obligation to pursue any course of business outlined in this document or any related presentation, or to develop or release any functionality mentioned therein. This document, or any related presentation, and SAP SE's or its affiliated companies' strategy and possible future developments, products, and/or platform directions and functionality are all subject to change and may be changed by SAP SE or its affiliated companies at any time for any reason without notice. The information in this document is not a commitment, promise, or legal obligation to deliver any material, code, or functionality. All forward-looking statements are subject to various risks and uncertainties that could cause actual results to differ materially from expectations. Readers are cautioned not to place undue reliance on these forward-looking statements, and they should not be relied upon in making purchasing decisions.

SAP and other SAP products and services mentioned herein as well as their respective logos are trademarks or registered trademarks of SAP SE (or an SAP affiliate company) in Germany and other countries. All other product and service names mentioned are the trademarks of their respective companies. See www.sap.com/copyright for additional trademark information and notices.