Employee Central Compound Employee API
# Content

1. **Introduction to the CompoundEmployee API.** ........................................ 6

2. **CompoundEmployee API: General Information About Full, Snapshot, and Delta Transmission Modes.** ............................................ 8
   2.1 Endpoints, WSDL, and State of Service. ........................................... 9
   2.2 Security and Session Management. ................................................... 9
   2.3 Thresholds and Limits of the CompoundEmployee API. ....................... 10
   2.4 Performance Recommendations for the CompoundEmployee API. .......... 11
   2.5 API Enhancements and Compatibility. .............................................. 12
   2.6 Audit Log and Metering Details. ...................................................... 13
   2.7 Setting Up the CompoundEmployee API. .......................................... 13
      Provisioning Settings for CompoundEmployee API. ............................ 13
      Granting Permissions for Full Access to the CompoundEmployee API. .... 16
      Granting Permissions for Restricted Access to the CompoundEmployee API. 17
      Granting Permissions for Segment Access to the CompoundEmployee API. 18
   2.8 Using the CompoundEmployee API. ................................................ 19
      Query and QueryMore. ....................................................................... 19
      Query Request. ................................................................................. 20
      Query Response. ............................................................................... 36
      API Metadata Access. ....................................................................... 73
      How the CompoundEmployee API Reacts to Data Purge. ..................... 75
      Error Handling in the CompoundEmployee API. ................................... 89

3. **Compound Employee API: Snapshot Mode.** ......................................... 99
   3.1 Data Purge Handling in Snapshot Mode. ............................................ 100

4. **CompoundEmployee API: Delta Transmission Mode.** .......................... 102
   4.1 Notation for Examples. ..................................................................... 103
   4.2 Delta Transmission Query Request. .................................................... 103
      SELECT Items. .................................................................................. 104
      Time-Restricted Delta Transmission Support for MDF Objects. ............ 105
      WHERE Expression. ......................................................................... 106
      Select Condition hiringNotCompleted. .............................................. 109
      Sorting by Start Date in the WHERE Expression. ............................... 112
      Controlling Parameters for Delta Transmission. ................................. 113
   4.3 Delta Transmission Query Response. ................................................ 116
      Query Response for Effective-Dated and Period-Based Delta Transmission. 116
   4.4 Action Codes. ................................................................................... 119

---

Employee Central Compound Employee API

Content
Learn about changes to the documentation for Employee Central CompoundEmployee API in recent releases.

2H 2021

<table>
<thead>
<tr>
<th>Type of Change</th>
<th>Description</th>
<th>More Info</th>
</tr>
</thead>
<tbody>
<tr>
<td>New</td>
<td>We added new information on permissions to the Compound Employee API.</td>
<td>Granting Permissions for Segment Access to the CompoundEmployee API [page 18]</td>
</tr>
<tr>
<td>Changed</td>
<td>We updated the information on the field non-unique external code in the MDF Picklists topic.</td>
<td>Picklists, Foundation Objects, and Generic Objects [page 44] Employee Central Application Message Error Codes [page 94]</td>
</tr>
</tbody>
</table>

1H 2021

<table>
<thead>
<tr>
<th>Type of Change</th>
<th>Description</th>
<th>More Info</th>
</tr>
</thead>
<tbody>
<tr>
<td>Changed</td>
<td>We added sample query code for Describe and DescribeEx.</td>
<td>Example: Describe and DescribeEx [page 165]</td>
</tr>
</tbody>
</table>

2H 2020

<table>
<thead>
<tr>
<th>What's New</th>
<th>Description</th>
<th>More Info</th>
</tr>
</thead>
<tbody>
<tr>
<td>We’ve added the Employee Central Compound Employee API: Delta Transmission Mode to this guide.</td>
<td>It’s located in the section called Employee Central Compound Employee API: Delta Transmission Mode.</td>
<td>CompoundEmployee API: Delta Transmission Mode [page 102]</td>
</tr>
<tr>
<td>New employment cost assignment segment</td>
<td>We’ve added a new segment EmpCostAssignment. You can use it to replicate public-sector-specific data like fund, fund center and grant to payroll systems.</td>
<td>Employee Cost Assignment [page 55] SELECT Clause [page 21] Query Response Structure [page 38]</td>
</tr>
<tr>
<td>What's New</td>
<td>Description</td>
<td>More Info</td>
</tr>
<tr>
<td>----------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>---------------------------------------------------------------------------</td>
</tr>
<tr>
<td>New segment in the Purge Status Object</td>
<td>We've added information about the <code>rehiredDateTime</code> attribute.</td>
<td>Structure of the Purge Status Overview Segment [page 76]</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Entities Supporting Purge Status Overview [page 79]</td>
</tr>
</tbody>
</table>
1 Introduction to the CompoundEmployee API

The CompoundEmployee application programming interface (API) for SAP SuccessFactors Employee Central is used to extract employee data out of Employee Central.

The CompoundEmployee API is based on the Simple Object Access Protocol (SOAP). It replicates employee master data from Employee Central to SAP systems, payroll systems, and benefits systems.

Key Features

The API returns all requested data of an employee in a single call providing a hierarchically structured response.

The query response comprises all time slices of all entities specified in the request. You can apply filters in the query request.

The API replicates only data that has been approved in Employee Central.

Only employees that have an employment are considered. Employees that don't have an employment, such as technical users aren't considered.

To improve the performance and due to the focus of the API, only those fields are extracted by the API that are needed for the respective replication process.

Supported Fields

Before you start implementing, please take the time to check which of the fields you need are supported by this API. We’re continually adding from release to release. You can also combine the CompoundEmployee API with other Employee Central APIs to cover any additional fields you need.

Snapshot Transmission Mode

In this mode, the API returns the data as it was on the snapshot date, considering all changes, corrections, and deletions.
Delta Transmission Mode

In this mode, the API only returns employee data that was created, changed, or deleted since the last replication.

The delta transmission mode of the CompoundEmployee API makes it possible to determine the concrete data changes of employees that have happened since a given point in time. In contrast to full transmission, consumers only get the changed data for an employee including the previous values, with a clear indication on how to process the data.
2 CompoundEmployee API: General Information About Full, Snapshot, and Delta Transmission Modes

Find out what the CompoundEmployee API is used for.

Employee Central can be used as leading application for employee master data. Third-party systems, such as payroll or benefits providers, require Employee Central employee data to trigger their own services and follow-up processes. Therefore, a regular replication of HR data to the third-party systems has to be ensured. For this purpose, replication processes call the CompoundEmployee API, a SOAP-based web service inside Employee Central, to retrieve employee master data out of Employee Central.

The CompoundEmployee API is a query API within Employee Central, which is used to extract the employee master data that is relevant for the third-party system. The CompoundEmployee API response is structured similarly to the data structures in Employee Central.

The CompoundEmployee API is commonly used to synchronize employee master data between Employee Central and other on-demand or on-premise applications. It returns data about new employees and about changes of employee data. Replication of employee master data by calling the CompoundEmployee API happens synchronously, which means that the response is immediately returned.
2.1 Endpoints, WSDL, and State of Service

Find out about endpoint and WSDL URLs of the CompoundEmployee API and about operations you can use to retrieve the API’s state of service.

Endpoint URLs

Your API endpoint depends on where your SAP SuccessFactors instance is located. It can be in one of several centers. The API uses the generic SFAPI web service endpoints for each data center location.

WSDL URLs

The generic WSDL can be retrieved by using the SFAPI URLs plus ?wsdl, for example: https://api.successfactors.eu/sfapi/v1/soap?wsdl

State of Service

An XML schema is provided that describes the XML response of the CompoundEmployee API including all substructures and elements. The XML schema is required for integration purposes.

The API provides the query and the queryMore operation, as well as the operations list, describe, and describeEx.

Related Information

Query and QueryMore [page 19]
List [page 74]
Describe and DescribeEx [page 74]

2.2 Security and Session Management

See how the CompoundEmployee API handles authentication and sessions.

Authentication is established through the Login operation. A successful login returns 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 session times out after 10 minutes of inactivity. You can also manually invalidate a session using the Logout method.

**Note**
SSO isn’t supported for SAP SuccessFactors APIs.

### 2.3 Thresholds and Limits of the CompoundEmployee API

To secure the stability of your integration, consider the thresholds and limits of the CompoundEmployee API.

**Maximum SOAP Message Size**

A maximum SOAP message size (HTTP content size) can’t exceed 5 MB. This is the limit when uploading binary attachments using SFAPI. In general, the total attachment storage size is controlled by the attachment storage configuration of the company instance.

**Maximum Number of Rows**

By default, a maximum of 200 rows is returned in a single `query` or `queryMore` method. This number can be set to a value from 1 to 800 by specifying the `maxRows` parameter in the query method. Reach best performance by setting the `maxRows` parameter to at least 400 or more.

In case you face issues because the amount of data to be returned is too high, you can reduce the number of rows for each `query` or `queryMore` call for your integration project. But remember that reducing this number leads to more `queryMore` operations, which in turn can slow down the transfer of data.

**Maximum Number of Employees**

Due to performance limitations, we recommend that you don’t retrieve more than 20,000 employees for each integration process run. If the number of employees to be extracted is greater than 20,000, you can limit the number of selected employees per run by using selection parameters such as `COMPANY`, `COMPANY_TERRITORY_CODE` (country/region), `PAY_GROUP`, or `EMPLOYEE_CLASS`.

**Maximum Number of Objects**

The maximum number of objects that can be exposed by the API is limited to 50,000 instances of an individual MDF object or composite subobject for each call. In case this number is exceeded, the API aborts the call and
shows an error message containing the object type in question and the limit applied. If this happens, we recommend that you reduce the number of objects to be returned by adjusting the filter conditions, such as `effective_end_date`, or reducing the page size of the query.

**Transferring Changed Employees**

Ensure that you don’t replicate employees many times although their data hasn’t changed. Use the `last_modified_on` parameter to transfer only employees that have changed since the last process run.

Please note that for `last_modified_on` queries, it’s only possible to select data changes for up to a maximum of three months into the past. Any date further in the past isn’t considered.

**Selecting Single Employees**

If you want to select multiple employees, don’t query them one by one. Avoid selecting single employees, in particular with a high request frequency.

A common use case here is to query data of the employees’ managers. Use the `associated_employee_information` segment to get the managers along with the employees, instead of querying this data separately.

### 2.4 Performance Recommendations for the CompoundEmployee API

Consider the following recommendations to secure the performance of the `CompoundEmployee` API.

**Scheduling API Calls**

Avoid triggering a large number of queries in parallel. Too many parallel queries can have a negative impact on performance and can put the stability of your integration at risk.

Don’t schedule API calls more frequently than every 15 minutes if they include the `last_modified_on` select parameter or request the segment `EmployeeDataReplicationElement`. In case certain updates (such as the termination of an employment) are needed earlier in the target system, consider consuming specific events to detect such changes immediately. For more information, see Setting Up Intelligent Services.
Minimizing Full Extraction Runs

API calls without a `last_modified_on` select parameter lead to a full extraction of employees, regardless of whether data has changed since the last replication run. We recommend that you only use such calls in initial load scenarios or when requesting the segment `EmployeeDataReplicationElement` in order to fetch entries from the Employee Central Data Replication Monitor.

An initial load should be a one-time activity rather than scheduled to be run regularly. In case you still need to schedule API calls without a `last_modified_on` parameter, make sure that they don’t run more often than once a day.

Mass Updates and Data Extraction Runs

Don’t carry out mass updates (for example, using imports) in parallel with a data extraction run performed by the API. Or else employees might not be picked up due to deferred storage on the database. Meaning that the data hasn’t yet been stored, but extraction is already running. In such a case, the `last_modified_on` time stamp could be older than the `last_modified_on` time stamp provided with the next query call, resulting in employees not being considered anymore.

2.5 API Enhancements and Compatibility

External applications consuming the `CompoundEmployee` API must consider that the API response message type can be enhanced with additional elements and attributes. The external application must be able to process the extended response successfully.

Additional elements and attributes can be supplied by SAP SuccessFactors and by customers, by enabling custom fields.

The API request message type can be enhanced by new processing and query string parameters, such as additional parameters for selecting or filtering data. Enhancements of request message types and parameters are always optional. The system doesn’t require the external application to provide optional values and parameter in the request.

XML element names and attribute names are always stable. Technical definitions of data types depend on your specific data model. This means that data types such as the field length can change. External applications can rely on XML element names and attribute names, but they can’t rely on data types. Use the describe or describeEx operations to retrieve a complete list of fields.
2.6 Audit Log and Metering Details

The SFAPI Audit Log is intended to help with support and debugging of API usage.

The SFAPI Audit Log allows you to inspect the API payload request made to the system and the corresponding API response sent by the system.

The SFAPI Audit Log is intended for developers, who use the API during an implementation, and for administrators, who can share information from this log with SAP SuccessFactors support. The tool allows you to download data from individual calls. You can then attach this data to a support incident to help resolve API-related issues.

⚠️ Caution

The SFAPI Audit Log allows access to potentially sensitive data. Restrict access to this tool to a limited number of trusted users.

The SFAPI Audit log captures payload details for the last 10,000 API calls. The API Metering Details page in the Audit Log gives you analytics on your API usage for the last 30 days. You can use this page to see API call history analytics, such as how many times the API was called or what was the total record counts accessed in your system.

2.7 Setting Up the CompoundEmployee API

What needs to be set up before you can start using the CompoundEmployee API.

Provisioning Settings for CompoundEmployee API [page 13]

Some settings are required in Provisioning for your SAP SuccessFactors instance to be able to use the CompoundEmployee API.

Granting Permissions for Full Access to the CompoundEmployee API [page 16]

Grant your API users the permissions they require for using the CompoundEmployee API.

Granting Permissions for Restricted Access to the CompoundEmployee API [page 17]

Grant your API users restricted access to specific data, such as employees in a specific country/region, when they use the CompoundEmployee API.

Granting Permissions for Segment Access to the CompoundEmployee API [page 18]

Grant your API users segment access to access only individual segments.

2.7.1 Provisioning Settings for CompoundEmployee API

Some settings are required in Provisioning for your SAP SuccessFactors instance to be able to use the CompoundEmployee API.

Making Company Settings in Provisioning [page 14]
Have the following settings be made in the Company Settings section in Provisioning to enable use of the CompoundEmployee API.

Making API Settings in Provisioning [page 15]
Have the following API settings be made in Provisioning to enable use of the CompoundEmployee API.

2.7.1.1 Making Company Settings in Provisioning

Have the following settings be made in the Company Settings section in Provisioning to enable use of the CompoundEmployee API.

Procedure

1. Log on to Provisioning for your SAP SuccessFactors instance.

   → Remember
   As a customer, you don’t have access to Provisioning. To complete tasks in Provisioning, contact your implementation partner. If you’re no longer working with an implementation partner, contact Product Support.

2. Select Company Settings.

3. Select the following checkboxes in the Web Services section:
   - SFAPI
   - Employee Central SOAP API

   The CompoundEmployee API is an Employee Central SOAP API and uses the SFAPI operations Login and Logout. Hence, enabling the API follows the standard process for SFAPI.

4. Search for Role-based permission and select the Role-based Permission (This will disable Administrative Domains) checkbox.

5. Enable the V2 UI by selecting the following checkboxes:
   - Employee Profile data audit
   - Effective Dated Data Platform and Enable Effective Dated fields in Basic Import
   - Employee Central V2 (i.e., Event Reason Derivation)
   - Enable youCalc rules engine for HRIS

6. If you want the API to provide information about global assignments, dependents in general, and dependents accompanying an employee on a global assignment, select the following checkboxes:
   - Enable Global Assignment Management
   - Enable Dependents Management

If you want the API to provide information about global assignment and dependents, you must also add the HRIS elements globalAssignmentInfo and personRelationshipInfo to the Succession Data Model. For more information, refer to the Succession Data Model (SDM) topic.
2.7.1.2 Making API Settings in Provisioning

Have the following API settings be made in Provisioning to enable use of the CompoundEmployee API.

Procedure

1. Log on to Provisioning for your SAP SuccessFactors instance.

   → Remember
   As a customer, you don’t have access to Provisioning. To complete tasks in Provisioning, contact your implementation partner. If you’re no longer working with an implementation partner, contact Product Support.

2. Select OData API Audit Log Setting in the API Settings & Tools section.

3. Select the Enable checkbox for Enable/Disable OData API Audit Log.

   Enable the data audit log for data compliance. Once it’s turned on, all employee data changes are recorded and you can see a history. If the audit log isn’t activated, the API raises an error message.

4. Select SFAPI Feature Settings in the API Settings & Tools section.

5. Choose the segments you want to include in the employee completeness check.

6. Select the checkbox for Enable compatibility mode (use only in alignment with SuccessFactors).

7. Select the Enable server paging checkbox.

   Server paging optimizes the paging of CompoundEmployee API query and queryMore calls.

8. Select the Disable “Data Replication Monitor” Update checkbox if you want to disable update of the Employee Central Data Replication Monitor by the CompoundEmployee API.

   Disabling the update makes sense if the monitor is delayed or there are too many asynchronous updates in the queue.


   The Zero Rows Detection algorithm determines if changes are available for last_modified_on and start_date selects. If the algorithm can’t find any changes, the data selects for the respective objects aren’t conducted, therefore improving performance.

10. Align with SAP SuccessFactors if you want to select the checkbox for Enable optimized data access (use only in alignment with SuccessFactors).

    Enabling this setting can cause performance degradation.

Related Information

Employee Completeness Check [page 41]
Query and QueryMore [page 19]
Select Parameters for the WHERE Clause [page 23]
2.7.2 Granting Permissions for Full Access to the CompoundEmployee API

Grant your API users the permissions they require for using the CompoundEmployee API.

Context

The permissions listed here give a user access to use the SAP SuccessFactors SFAPI. SFAPI access includes access to the CompoundEmployee API. Please note that the permissions grant only API access. They don’t grant access to the SAP SuccessFactors UI.

i Note

Either grant the Employee Central HRIS SOAP API permission together with the SFAPI User Login permission, to give the API users unrestricted access to the CompoundEmployee API. Or grant the Employee Central Compound Employee API (restricted access) permission together with the SFAPI User Login permission, to restrict the access to data of specific employees. Don’t select both permissions, Employee Central HRIS SOAP API and Employee Central Compound Employee API (restricted access), for the same permission role.

Procedure

1. Go to User Permissions and select the General User Permission category.
2. Select the checkbox for the SFAPI User Login permission.

   The CompoundEmployee API is an Employee Central SOAP API and uses the SFAPI operations Login and Logout. Hence, providing API login permissions follows the standard process for SFAPI.

3. Go to Administrator Permissions and select the Employee Central API category.
4. Select the checkbox for the Employee Central HRIS SOAP API permission.

Next Steps

You can test that the API works by using any commonly available tools, such as the SOAP UI tool.

Related Information

List of Role-Based Permissions
2.7.3 Granting Permissions for Restricted Access to the CompoundEmployee API

Grant your API users restricted access to specific data, such as employees in a specific country/region, when they use the CompoundEmployee API.

Context

The permissions listed here give a user restricted access to the CompoundEmployee API, so that they can extract only specific data.

<table>
<thead>
<tr>
<th>i Note</th>
</tr>
</thead>
<tbody>
<tr>
<td>Either grant the Employee Central HRIS SOAP API permission together with the SFAPI User Login permission, to give the API users unrestricted access to the CompoundEmployee API. Or grant the Employee Central Compound Employee API (restricted access) permission together with the SFAPI User Login permission, to restrict the access to data of specific employees. Don’t select both permissions, Employee Central HRIS SOAP API and Employee Central Compound Employee API (restricted access), for the same permission role.</td>
</tr>
</tbody>
</table>

Procedure

1. Go to User Permissions and select the General User Permission category.
2. Select the checkbox for the SFAPI User Login permission.
3. Go to Administrator Permissions and select the Employee Central API category.
4. Select the checkbox for the Employee Central Compound Employee API (restricted access) permission.

This permission restricts the accessible data according to the definition of the target population, for example, for a country/region or a department.

<table>
<thead>
<tr>
<th>i Note</th>
</tr>
</thead>
<tbody>
<tr>
<td>If the number of returned employees is less than the requested maximum number of employees, this doesn’t mean that this package is the last one. If the result field hasMore is set to value true, a consumer of the API has to perform queryMore operations until hasMore returns the value false.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>→ Remember</th>
</tr>
</thead>
<tbody>
<tr>
<td>If the last modification time of a specific employee’s data is before the given last_modified_on select parameter, this employee isn’t returned in the response, even if the employee is part of the target population. This mismatch is caused by the asynchronous update of the list of employees contained in RBP permission groups. Updating the employee list of a permission group doesn’t change the data of the employees in question. That’s why no new last_modified_on date is set for these employees.</td>
</tr>
</tbody>
</table>
2.7.4 Granting Permissions for Segment Access to the CompoundEmployee API

Grant your API users segment access to access only individual segments.

Prerequisites

You have the permission to access the Employee Central HRIS SOAP API or the Employee Central Compound Employee API (restricted access).

Context

The Employee Central HRIS SOAP API permission or the Employee Central Compound Employee API (restricted access) permission can be further restricted, by assigning the Employee Central Compound Employee API (segment access) permission.

The Employee Central Compound Employee API (segment access) allows the user with this role to access segments according to the configuration set up only. The configuration is set up by choosing the option Others and selecting segments, which the user should be able to access.

There are 2 options for granting access to Employee Central Compound Employee API segments:

Option All provides full access to Compound Employee API segments, since all segments remain accessible.

Option Others provides the possibility to grant access only to defined segments. At least one segment must be selected.

i Note

The segment person isn’t displayed in the list as it is accessible by default.

Procedure

1. Go to User Permissions and select the General User Permission category.
2. Select the checkbox for the SFAPI User Login permission.
3. Go to Administrator Permissions and select the Employee Central API category.
4. Select the checkbox for the *Employee Central HRIS SOAP API (segment access)* permission.

## 2.8 Using the CompoundEmployee API

The CompoundEmployee API selects multiple entities related to an employee and returns all data in a hierarchically-structured response XML with the employee person data as the root node. Find out how this works in detail.

**Query and QueryMore [page 19]**
- A typical retrieval sequence consists of a login call, a CompoundEmployee API call, and a number of subsequent queryMore API calls.

**Query Request [page 20]**
- What you should know about the CompoundEmployee API query request.

**Query Response [page 36]**
- What you should know about the CompoundEmployee API query response.

**API Metadata Access [page 73]**
- The CompoundEmployee API also provides operations that return its metadata.

**How the CompoundEmployee API Reacts to Data Purge [page 75]**
- The CompoundEmployee API provides some optional data purge checks, which can be enabled by the consumers.

**Error Handling in the CompoundEmployee API [page 89]**
- How the CompoundEmployee API handles error situations.

### 2.8.1 Query and QueryMore

A typical retrieval sequence consists of a login call, a CompoundEmployee API call, and a number of subsequent queryMore API calls.

The API query call specifies the entities to be returned plus the selection criteria. This call returns a limited number of employees. To retrieve the remaining records, you need to issue a series of queryMore API calls until all employees are retrieved. In each call, queryMore you need to specify the session ID provided by the first query response.

The retrieval sequence returns all employees fulfilling the selection criteria specified in the request according to the following rules:

- Only employees are considered that have successfully completed the approval process.
- Technical users and other persons without employment data aren’t selected by the API.

**Note**

For each employee that fulfills the selection criteria, all elements specified in the SELECT clause are returned regardless of whether they fulfill the criteria individually.
2.8.2 Query Request

What you should know about the CompoundEmployee API query request.

**Query Request Structure [page 20]**
- The CompoundEmployee API query request is made up of a SELECT clause, FROM clause, and WHERE clause.

**SELECT Clause [page 21]**
- The SELECT clause of the CompoundEmployee API contains a list of all entities to be returned as part of the hierarchical query result XML. The restriction of fields isn’t supported.

**WHERE Clause [page 22]**
- The WHERE clause in the CompoundEmployee API query request specifies the conditions that employees need to fulfill in order to be included in the query result.

**Overview of Query Parameters [page 28]**
- The <param> tag contains parameters that control the behavior of the CompoundEmployee API.

**Server Paging [page 35]**
- Server paging optimizes the paging of CompoundEmployee API query and queryMore calls.

### 2.8.2.1 Query Request Structure

The CompoundEmployee API query request is made up of a SELECT clause, FROM clause, and WHERE clause.

Let’s look at an example of what the query request can look like:

```xml
<query>
  <queryString>
    SELECT person, personal_information, employment_information, job_information
    FROM CompoundEmployee
    WHERE last_modified_on > to_datetime ('YYYY-MM-DDTHH:MM:SSZ')
  </queryString>
  <param>
    <name>maxRows</name>
    <value>50</value>
  </param>
</query>
```

The query request of the API call has the following properties:

<table>
<thead>
<tr>
<th>Property</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>param</td>
<td>List of optional parameters (supported parameters are startingRow, maxRow, and traceLevel)</td>
</tr>
<tr>
<td>queryString</td>
<td>Select statement following an SQL-like syntax</td>
</tr>
</tbody>
</table>

**Note**

<select_items> and <expression> aren’t case-sensitive and are considered as long they’re spelled correctly.
Related Information

Example: Sample Query Requests [page 167]

2.8.2.2 SELECT Clause

The SELECT clause of the CompoundEmployee API contains a list of all entities to be returned as part of the hierarchical query result XML. The restriction of fields isn’t supported.

The following substructures are supported in the SELECT clause:

- 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
- BenefitsIntegrationOneTimeInfo
- BenefitsIntegrationRecurringInfo
- EmpCostAssignment
- 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
- SecondaryAssignments
2.8.2.3 WHERE Clause

The WHERE clause in the CompoundEmployee API query request specifies the conditions that employees need to fulfill in order to be included in the query result.

The WHERE clause consists of subexpressions, which can be combined by the logical operator AND. A subexpression is made up of field, operator, and value or values.

```
WHERE company = 'SAP' AND employee_class IN ('C','P')
```

The CompoundEmployee API doesn't support expressions with complex conditions on several fields and different logical operators.

The following behavior must be considered by the consumer:

- Expressions within the WHERE clause can only be combined by the logical operator AND.
- The effective_end_date condition can only be compared with the value following the pattern `to_date(<date value>)` in format YYYY-MM-DD. No other date formats are possible here.
- Conditions related to effective dated fields such as employee_class, department, division, or company_territory_code can lead to unexpected results when they're used in conjunction with a condition containing the field effective_end_date. The reason is that effective_end_date is used as a filter criterion rather than as a select condition in the SQL sense. That means that an employee can be contained in the result set because the specified employee_class is contained in one of the time slices. But the time slice can be filtered out by the effective_end_date condition.

Select Parameters for the WHERE Clause [page 23]

The WHERE clause of the CompoundEmployee API supports a list of select parameters.

Effective End Date Filter [page 27]

The EFFECTIVE_END_DATE parameter in the WHERE clause of the CompoundEmployee API plays a special role, since it isn’t directly applied when selecting the data but is used to filter the result.

Sorting by Start Date [page 27]

The CompoundEmployee API supports ascending and descending sorting by start date for effective-dated entities.
2.8.2.3.1 Select Parameters for the WHERE Clause

The `WHERE` clause of the CompoundEmployee API supports a list of select parameters.

### Supported Parameters

<table>
<thead>
<tr>
<th>Field</th>
<th>Remark</th>
<th>Valid Operators</th>
</tr>
</thead>
<tbody>
<tr>
<td>LAST_MODIFIED_ON</td>
<td>Returns all employees, for which any employee data has changed since this date and time. The modification date applies on all effective dated records. The date can only go back as far as 3 months. User information isn’t considered for queries by last_modified_on. Deletion of records is considered in queries for last modified employees. These deleted records aren’t returned by the API. For example: WHERE last_modified_on &gt; to_datetime('2012-11-06T23:30:00Z') WHERE last_modified_on &gt; to_date('01/07/10', 'DD/MM/YY') &gt;</td>
<td>=, &gt;, &gt;=, &lt;, &lt;=</td>
</tr>
<tr>
<td>START_DATE</td>
<td>Returns all employees for which any employee data is effective from this date on.</td>
<td>&gt;=</td>
</tr>
<tr>
<td>COMPANY_TERRITORY_CODE</td>
<td>Returns all employees that have a job at a company located in the provided country/region at any point in time. The companies are determined via table FO_LEGAL_ENTITY_T.</td>
<td>=, IN</td>
</tr>
<tr>
<td>PERSON_ID</td>
<td>Returns all employees with the respective PERSON_ID.</td>
<td>=, IN, NOT IN</td>
</tr>
<tr>
<td>PER_PERSON_UUID</td>
<td>Returns all employees with the respective PER_PERSON_UUID.</td>
<td>=, IN, NOT IN</td>
</tr>
<tr>
<td>PERSON_ID_EXTERNAL</td>
<td>Returns all employees with the respective PERSON_ID_EXTERNAL.</td>
<td>=, IN, NOT IN</td>
</tr>
<tr>
<td>USER_ID</td>
<td>Returns all employees with the respective USER_ID.</td>
<td>=, IN, NOT IN</td>
</tr>
<tr>
<td>COMPANY</td>
<td>Returns all employees that have a job at the selected company. The select is based on the external code of the company.</td>
<td>=, IN</td>
</tr>
<tr>
<td>EMPLOYEE_CLASS</td>
<td>Returns all employees that have a job in the respective employee class. The selection is based on the external code of the employee class.</td>
<td>=, IN</td>
</tr>
<tr>
<td>Field</td>
<td>Remark</td>
<td>Valid Operators</td>
</tr>
<tr>
<td>------------------</td>
<td>------------------------------------------------------------------------</td>
<td>-----------------</td>
</tr>
<tr>
<td>DEPARTMENT</td>
<td>Returns all employees that have a job in the provided department. The select is based on the external code of the department.</td>
<td>=, IN</td>
</tr>
<tr>
<td>DIVISION</td>
<td>Returns all employees that have a job in the provided division. The select is based on the external code of the division.</td>
<td>=, IN</td>
</tr>
<tr>
<td>BUSINESS_UNIT</td>
<td>Returns all employees that have a job in the provided business unit. The select is based on the external code of the business unit.</td>
<td>=, IN</td>
</tr>
<tr>
<td>LOCATION</td>
<td>Returns all employees that have a job in the provided location. The select is based on the external code of the location.</td>
<td>=, IN</td>
</tr>
<tr>
<td>JOB_CODE</td>
<td>Returns all employees that have a job with the provided job code. The select is based on the external code of the job code.</td>
<td>=, IN</td>
</tr>
<tr>
<td>PAY_GROUP</td>
<td>Returns all employees that have a job with the provided pay group. The select is based on the external code of the pay group.</td>
<td>=, IN</td>
</tr>
<tr>
<td>COMPENSATION_PAY_GROUP</td>
<td>Returns all employees that have a compensation with the provided pay group. The select is based on the external code of the pay group.</td>
<td>=, IN</td>
</tr>
<tr>
<td>SNAPSHOT_DATE</td>
<td>Returns the employee data for a certain point in time. The date can only go back as far as 3 months.</td>
<td>=</td>
</tr>
<tr>
<td>SourceOfRecord</td>
<td>Returns all employees that have an employment with the provided source of record information. The select is based on the external code of the source of record information.</td>
<td>=, IN</td>
</tr>
</tbody>
</table>

**i Note**

You have to enable and make the field `sourceOfRecord` visible in the Succession Data Model before this filter parameter can be used. Additionally, you have to assign the relevant picklist to the field in the data model.
<table>
<thead>
<tr>
<th>Field</th>
<th>Remark</th>
<th>Valid Operators</th>
</tr>
</thead>
<tbody>
<tr>
<td>isContingentWorker</td>
<td>Returns all employees that have an employment with the given value in field IsContingentWorker. The absence of the isContingentWorker condition is treated as isContingentWorker=false to preserve compatibility with older integration processes that expect real employees only. In order to select all persons that are contingent workers, use the condition isContingentWorker=true. If you want to request only regular employees, use the condition isContingentWorker=false. If you want both employment types, use condition isContingentWorker IN (true, false). For more information about how to set up and configure Employee Central for contingent workers, refer to Contingent Workforce Management.</td>
<td>=, IN</td>
</tr>
<tr>
<td>fromDate...toDate</td>
<td>Selects employees that have changes that become effective in the given time period. Additionally, the time period is applied as a filter to all effective dated segments so that only time slices are returned that intersect with the given time period.</td>
<td>=</td>
</tr>
<tr>
<td>selectFromDate</td>
<td>Returns all employees that satisfy all job and/or compensation conditions where the effective end date of the respective time slices is greater than or equal to the value of selectFromDate. The value must be a fixed date format, for example to_date('2016-01-01','yyyy-MM-dd').</td>
<td>=</td>
</tr>
<tr>
<td>selectToDate</td>
<td>Returns all employees that satisfy all job and/or compensation conditions where the effective start date of the respective time slice is lower than or equal to the value selectToDate. The value must be a fixed date format, for example to_date('2016-01-01','yyyy-MM-dd').</td>
<td>=</td>
</tr>
</tbody>
</table>
### Effective Period Selection

Using the select parameters `selectFromDate` and `selectToDate` you can further restrict the set of selected employees either by specifying an open period, by using only one of the parameters or by using both parameters to determine a fixed time range.

**i Note**

You can only use effective period selection in combination with other select parameters of `job_information` or `compensation_information`. If you don’t use an additional parameter, the query returns an error.
2.8.2.3.2 Effective End Date Filter

The **EFFECTIVE_END_DATE** parameter in the **WHERE** clause of the CompoundEmployee API plays a special role, since it isn’t directly applied when selecting the data but is used to filter the result.

<table>
<thead>
<tr>
<th>Filter Parameter</th>
<th>Remark</th>
<th>Valid Operations</th>
</tr>
</thead>
<tbody>
<tr>
<td>EFFECTIVE_END_DATE</td>
<td>The condition is applied as a filter to all segments of the employee that are effective-dated. Depending on the operator, the following results are returned:</td>
<td>=: The query returns the entry of an effective-dated segment, which is valid on the given date (\text{EFFECTIVE_START_DATE} \leq \text{given date} \leq \text{EFFECTIVE_END_DATE}). (&gt;&lt;=): The query returns all occurrences of an effective-dated segment that are valid on the given date or any later date (\text{given date} \leq \text{EFFECTIVE_END_DATE}).</td>
</tr>
</tbody>
</table>

**Note**

The condition for the **EFFECTIVE_END_DATE** parameter is more a filter than a selection condition. Conditions on all the other elements determine if an employee is to be included in the result. Once an employee fulfills these conditions in any time slice, all time slices of all elements specified in the **SELECT** clause are included in the response, regardless of whether an individual element fulfills a condition. The condition on the **EFFECTIVE_END_DATE** parameter, however, determines whether an individual element is included in the response or not.

2.8.2.3.3 Sorting by Start Date

The CompoundEmployee API supports ascending and descending sorting by start date for effective-dated entities.

Time slices of effective-dated objects in the response are by default sorted in descending order by start date. This means the newest time slice is always the first in the response. Semantic keys and sequence numbers are also considered in sorting. To change the sorting order, add the following expression to the **WHERE** clause:

```xml
<queryString>
  SELECT person, personal_information, address_information, employment_information, job_information
  FROM CompoundEmployee
  ORDER BY start_date ASC / DESC
</queryString>
```
Syntax of ORDER BY Expression

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Sorting Order</th>
</tr>
</thead>
<tbody>
<tr>
<td>ORDER BY start_date DESC</td>
<td>descending</td>
</tr>
<tr>
<td>ORDER BY start_date ASC</td>
<td>ascending</td>
</tr>
<tr>
<td>ORDER BY start_date</td>
<td>ascending</td>
</tr>
</tbody>
</table>

Related Information

Start Date Condition [page 59]

2.8.2.4 Overview of Query Parameters

The `<param>` tag contains parameters that control the behavior of the CompoundEmployee API.

Parameter Tag

Each `<param>` tag consists of a parameter name and one or more values.

```xml
<urn:param>
  <urn:name>parameter name</urn:name>
  <urn:value>parameter values</urn:value>
</urn:param>
```

Parameter values are case insensitive and are considered as long as the spelling is correct.

Supported Parameters

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Use To...</th>
<th>Parameter Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>maxRows</td>
<td>Define the maximum number of rows to be returned by the call.</td>
<td></td>
</tr>
<tr>
<td>Parameter Name</td>
<td>Use To...</td>
<td>Parameter Values</td>
</tr>
<tr>
<td>---------------------</td>
<td>---------------------------------------------------------------------------</td>
<td>----------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>startingRow</td>
<td>Define the number of the first row to be returned by the call.</td>
<td>For example, if startingRow is 3, the first 2 rows are skipped.</td>
</tr>
<tr>
<td>queryMode</td>
<td>Switch between full and delta transmission, or to enable snapshot mode.</td>
<td>snapshot: Switch on the snapshot mode. This option is available for delta transmission. If you’re interested, contact SAP Support. delta: Switch on delta transmission. periodDelta: Enable period-based delta transmission.</td>
</tr>
<tr>
<td>externalKeyMapping</td>
<td>Return object keys that originate from an external system.</td>
<td>costCenter: If specified, the API returns the value of field EXTERNAL_OBJECT_ID of the cost center MDF object instead of the external code.</td>
</tr>
<tr>
<td>costCenter</td>
<td>If specified, the API returns the value of field EXTERNAL_OBJECT_ID of the cost center MDF object instead of the external code</td>
<td>Only for the delta mode</td>
</tr>
<tr>
<td>Parameter Name</td>
<td>Use To...</td>
<td>Parameter Values</td>
</tr>
<tr>
<td>---------------------------</td>
<td>---------------------------------------------------------------------------</td>
<td>--------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>resultOptions</td>
<td>Switch between different options on how much data is to be returned in the response</td>
<td>allJobChangesPerDay: If specified, the API returns all Job Information time slices that result from multiple job changes per day. Normally, in case there are several Job Information time slices with the same start date, the API doesn’t return all time slices. It only returns the Job Information time slice with the highest sequence number because this time slice contains the valid business data. If the parameter value is specified, the consumer gets all Job Information time slices, thus knowing about the historic changes and events that triggered each of the changes. The parameter value is only supported with the full transmission and the snapshot mode.</td>
</tr>
<tr>
<td>allCompensationChangesPerDay</td>
<td>If specified, the API returns all Compensation Information time slices that result from multiple compensation changes per day. As a result, the consumer can see historic changes and the events that triggered these changes. This option is only supported with the full transmission and the snapshot mode.</td>
<td></td>
</tr>
<tr>
<td>relevantEmployeesOnly</td>
<td>relevantEmployeesOnly: If specified, the API excludes all employees that don’t have relevant data. This option can have an impact on paging, since the result may have fewer employees than specified using the maxRows parameter.</td>
<td></td>
</tr>
<tr>
<td>Parameter Name</td>
<td>Use To...</td>
<td>Parameter Values</td>
</tr>
<tr>
<td>-----------------------------</td>
<td>---------------------------------------------------------------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>configuredFieldsOnly</td>
<td></td>
<td>configuredFieldsOnly: If specified, the API only returns fields that are set to visible in the data model. Note that DescribeEx returns all fields anyway.</td>
</tr>
<tr>
<td>Parameter Name</td>
<td>Use To...</td>
<td>Parameter Values</td>
</tr>
<tr>
<td>------------------------</td>
<td>--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>segmentsForEnhanced</td>
<td>Return multiple events performed for the Job Information and Compensation Information segments on the same day when <code>effective_end_date</code> is after that day. Normally, only the latest event, which has the highest sequence number, is returned by the API in such a case. Because only this event has a time slice that overlaps with <code>effective_end_date</code>. Other events performed on the same day only have a time stamp, which is before <code>effective_end_date</code> and therefore isn’t considered. The parameter is only supported with the full transmission and the snapshot mode.</td>
<td></td>
</tr>
<tr>
<td>EffectiveEndDateFilter</td>
<td></td>
<td>job_information: If specified, the API returns all Job Information time slices that result from multiple job changes for each day. If the parameter value is specified, the consumer gets the time slice that overlaps with <code>effective_end_date</code> as well as all other time slices that have changes on the same day. The consumer can then react on all changes and all events that triggered the changes. The parameter value works only when the <code>effective_end_date</code> parameter is included in the query request and the <code>resultOptions</code> parameter has the value <code>allJobChangesPerDay</code>.</td>
</tr>
</tbody>
</table>

compensation_information: If specified, the API returns all Compensation Information time slices that result from multiple compensation changes for each day. If the parameter value is specified, the consumer gets the time slice that overlaps with `effective_end_date` as well as all other time slices that have changes on the same day. The consumer can then react on all changes and all events that triggered the changes. The parameter value works only when the `effective_end_date` parameter is included in the query request and the `resultOptions` parameter has the value `allCompensationChangesPerDay`. |
<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Use To...</th>
<th>Parameter Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>segmentsForCompleteness</td>
<td>Exclude employees that don’t have a complete set of master data from the response. You can define what the complete set of master data is in Provisioning. This process-specific parameter overrules the company-specific and process-un-specific Provisioning setting <code>Include in employee completeness check</code>. An error is raised for employees whose data in the following segments is incomplete:&lt;br&gt;  - <code>personal_information</code>&lt;br&gt;  - <code>job_information</code>&lt;br&gt;  - <code>compensation_information</code></td>
<td>“” or “”: If specified, the Provisioning settings are overruled and no errors are raised for employees with incomplete master data.</td>
</tr>
<tr>
<td>triggerReplicationMonitorUpdate</td>
<td>Trigger Data Replication Monitor updates by the CompoundEmployee API.</td>
<td></td>
</tr>
<tr>
<td>purgeOptions</td>
<td>Enable additional validations for Data Retention Management and Data Purging.</td>
<td><code>validateEffectiveEndDateFilter</code>: If specified, errors are raised for all employees, for whom the effective end date filter is within a purge period for one or more of the selected segments. The data of these employees isn't returned by the API.</td>
</tr>
</tbody>
</table>
**Parameter Name**
suppressUnwantedGlobalInfo

**Use To...**
Filter out Global Information within Personal Information based on the country/region. To comply with data protection and privacy regulations, you can set this parameter so that only those parts of Global Information are transferred to the downstream system that are required there.

**Parameter Values**
yes: If specified, Global Information is filtered for the country/region or countries/regions selected with the employment_information_country or the employment_information_company parameter in the WHERE clause.

**Example**
An ERP system holding information about German employees must get Global Information only for Germany. Global Information for other countries/regions must not be transferred to this ERP system.

The parameter works only when the employment_information_country or the employment_information_company parameter is included in the WHERE clause of the query request. If both parameters, employment_information_country and employment_information_company are selected, only employment_information_country is considered.

The filtering applies to the following segments:
- personal_information
- dependent_information(dependent_personal_information)
- national_id_card

If these segments aren’t included, the parameter doesn’t have any effect.

The filtering applies to all time slices.

The parameter is only supported with the full transmission mode.

`
<urn:query>
<urn:queryString>select person,personal_information,employment_information,job_information,global_assignment_information,payment_information from CompoundEmployee where employment_information_country = 'DEU' and employment_information_company = 'ACE_DEU'
</urn:queryString>
</urn:query>
`
### Parameter Name
### Use To...

- **Suppress Unwanted Global Information**
  - `suppressUnwantedGlobalInfo=yes`
  - If specified, the API returns only changed segments with an action code not equal to NO_CHANGE in delta transmission.

- **Changed Segments Only**
  - If specified, the API returns only changed segments in delta transmission. You have to combine this parameter value with the `changedSegmentsOnly` parameter value.

- **Changed Fields Only**
  - If specified, the API returns only changed fields of a changed segment in delta transmission. You have to combine this parameter value with the `changedSegmentsOnly` parameter value.

- **Is Not First Query**
  - The parameter value is only supported in period-based delta transmission and in combination with the `fromDate` and `toDate` select parameter.

- **Extended Consistency Check**
  - If specified during delta transmission, the API conducts additional validations for dates of time slices. It throws an error for affected employees in cases of overlapping time slices or gaps between time slices where no gaps are allowed. For the affected employees, a log entry is returned that has the severity ERROR.

- **Render Previous Tags**
  - If specified, the way that previous values are exposed in the response message is changed. The API returns previous values in a separate tag and no longer in mixed data types.

### 2.8.2.5 Server Paging

Server paging optimizes the paging of CompoundEmployee API query and queryMore calls.

With server paging, the whole result set of employees fulfilling the `WHERE` condition is determined only once in the CompoundEmployee API query operation. The result is buffered and reused in every subsequent `queryMore` operation. In contrast, in standard paging the result set is calculated in every `queryMore` operation over and over again. As a result, server paging increases performance and avoids overlaps between the results of subsequent query calls.

For reasons of security, the buffer size is limited to 1,000,000 records for each HTTP session. If this limit is reached and a new query is executed, standard paging is used instead of server paging.

#### Note

- Server paging must be enabled in Provisioning.
Related Information

Provisioning Settings for CompoundEmployee API [page 13]

2.8.3 Query Response

What you should know about the CompoundEmployee API query response.

Query Response Structure [page 38]
The CompoundEmployee API query response has a defined list of elements and follows the current Employee Central data model to build a hierarchically-structured XML.

Employee Completeness Check [page 41]
Use the employee completeness check to filter out employees whose master data isn't complete in the CompoundEmployee API query response.

Performance-Improved XML Rendering [page 42]
The CompoundEmployee API uses the performance-improved XML rendering.

Compatibility Mode [page 43]
If you use the performance-improved XML rendering and experience incompatibilities, you can enable the compatibility mode for the CompoundEmployee API.

Dynamic Field List [page 44]
The CompoundEmployee API returns a hybrid field list contained in the query response.

Picklists, Foundation Objects, and Generic Objects [page 44]
See how the CompoundEmployee API handles picklists, foundation objects, and generic objects.

Date/Time Handling [page 45]
The CompoundEmployee API query expressions handle date and time in different ways, depending on whether the last_modified_on or the effective_end_date parameter is used.

Logon User ID and User Characteristics [page 46]
The CompoundEmployee API query response holds information about the logon user.

Employee Central Entity Semantic Keys [page 46]
Some Employee Central entities have semantic keys that are used in the CompoundEmployee API query response to identify and distinguish records of said entities and include the parent segment ID.

Country/Region-Specific Fields [page 47]
For the country/region-specific fields in the segments job information, compensation information, and employment information external code values of picklists, foundation objects, and generic object instances are returned by the CompoundEmployee API. No matter whether the model is active or not, as long as there's data in the database for those specific fields.

External Key Mapping for Cost Centers [page 49]
The CompoundEmployee API supports external key mapping for cost centers.

Global Assignments [page 49]
The CompoundEmployee API returns global assignment information in the query response.
The CompoundEmployee API query response can also contain information about secondary assignments.

**Payment Information [page 50]**
The CompoundEmployee API returns MDF-based Payment Information data in the query response.

**Person Relationship and Employee Dependents [page 51]**
The CompoundEmployee API can return the employees’ dependents and also their person relationships in the query response.

**Emergency Contact [page 53]**
Enable the emergency_contact_primary segment in the CompoundEmployee API query response by adding the emergency_contact_primary select item to the select statement.

**Associated Employee Information [page 53]**
To get associated employee information, specify the selection items employment_information and associated_employee_information along with job_information or job_relation in the query.

**One-Time and Recurring Information for Benefits Integration [page 54]**
Find out about the benefits integration segments of the CompoundEmployee API, which can be used to transfer amounts such as insurance premiums or contributions to savings plans to a payroll system.

**Employee Cost Assignment [page 55]**
Find out about the Employee Cost Assignment segment of the CompoundEmployee API, which can be used to replicate business information like funds and grants to a payroll system.

**Income Tax Declaration [page 56]**
The Income Tax Declaration object is supported in the CompoundEmployee API in full transmission mode and can be enabled by including the select parameter ItDeclaration in the select items.

**Personal Documents [page 57]**
The CompoundEmployee API query returns information about employee documents, but not the document itself.

**Contingent Worker Employment Filter [page 57]**
The CompoundEmployee API can filter contingent workforce employments and regular employments of an employee.

**hiringNotCompleted Employment Filter [page 58]**
Use the hiringNotCompleted parameter in the query request of the CompoundEmployee API if you want your downstream systems to receive only data that has been validated against the employee data model in Employee Central.

**assignment_class Employment Filter [page 58]**
Use the assignment_class parameter in the query request if you want to filter employments for their assignment type such as standard employment or global assignment.

**Start Date Condition [page 59]**
Use the START_DATE parameter of the CompoundEmployee API when you want to receive employee data because the employee has time slices starting after a specific date.

**Extensibility of the CompoundEmployee API [page 70]**
The CompoundEmployee API supports two forms of extensibility: Employee Central custom fields and custom MDF objects.
2.8.3.1 Query Response Structure

The CompoundEmployee API query response has a defined list of elements and follows the current Employee Central data model to build a hierarchically-structured XML.

Elements of the Query Response

<table>
<thead>
<tr>
<th>Element</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>hasMore</td>
<td>Indicator if more results are available.</td>
</tr>
<tr>
<td>numResults</td>
<td>Number of top-level results.</td>
</tr>
<tr>
<td>querySessionId</td>
<td>Query session ID for paging.</td>
</tr>
<tr>
<td>sfobject</td>
<td>List of employees with subentities, returned as hierarchical XML.</td>
</tr>
</tbody>
</table>

Structured XML Built by the Query Response

<table>
<thead>
<tr>
<th>Segments and Subsegments</th>
<th>Cardinality</th>
<th>Effective-Dated</th>
</tr>
</thead>
<tbody>
<tr>
<td>person</td>
<td>0..n</td>
<td>-</td>
</tr>
<tr>
<td>personal_information</td>
<td>0..n</td>
<td>Yes</td>
</tr>
<tr>
<td>Personal_information_{count}</td>
<td>0..1*</td>
<td>Yes</td>
</tr>
<tr>
<td>address_information</td>
<td>0..n</td>
<td>Yes</td>
</tr>
<tr>
<td>email_information</td>
<td>0..n</td>
<td>No</td>
</tr>
<tr>
<td>phone_information</td>
<td>0..n</td>
<td>No</td>
</tr>
<tr>
<td>person_relation</td>
<td>0..n</td>
<td>Yes</td>
</tr>
<tr>
<td>employment_information</td>
<td>0..n</td>
<td>No</td>
</tr>
<tr>
<td>global_assignment_informati</td>
<td>0..n</td>
<td>No</td>
</tr>
<tr>
<td>job_information</td>
<td>0..n</td>
<td>Yes</td>
</tr>
<tr>
<td>alternative_cost_distributi</td>
<td>0..n</td>
<td>Yes</td>
</tr>
<tr>
<td>alternative_cost_distributi_{item}</td>
<td>0..n</td>
<td>Yes</td>
</tr>
<tr>
<td>compensation_information</td>
<td>0..n</td>
<td>Yes</td>
</tr>
<tr>
<td>paycompensation_recurring</td>
<td>0..n</td>
<td>Yes</td>
</tr>
<tr>
<td>Segments and Subsegments</td>
<td>Cardinality</td>
<td>Effective-Dated</td>
</tr>
<tr>
<td>------------------------------------------------</td>
<td>-------------</td>
<td>-----------------</td>
</tr>
<tr>
<td>paycompensation_non_recurring</td>
<td>0..n</td>
<td>No</td>
</tr>
<tr>
<td>payment_information</td>
<td>0..n</td>
<td>Yes</td>
</tr>
<tr>
<td>accompanying_dependent</td>
<td>0..n</td>
<td>Yes</td>
</tr>
<tr>
<td>job_relation</td>
<td>0..n</td>
<td>Yes</td>
</tr>
<tr>
<td>deduction_recurring</td>
<td>0..n</td>
<td>Yes</td>
</tr>
<tr>
<td>deduction_recurring_item</td>
<td>0..n</td>
<td>Yes</td>
</tr>
<tr>
<td>deduction_non_recurring</td>
<td>0..n</td>
<td>No</td>
</tr>
<tr>
<td>ItDeclaration</td>
<td>0..n</td>
<td>Yes</td>
</tr>
<tr>
<td>PaymentInformationV3</td>
<td>0..n</td>
<td>Yes</td>
</tr>
<tr>
<td>SecondaryAssignmentPeriod***</td>
<td>0..n</td>
<td>Yes</td>
</tr>
<tr>
<td>associated_employee_information</td>
<td>0..n</td>
<td>No</td>
</tr>
<tr>
<td>associated_employee_employment_information</td>
<td>0..n</td>
<td>No</td>
</tr>
<tr>
<td>associated_employee_job_information</td>
<td>0..n</td>
<td>Yes</td>
</tr>
<tr>
<td>BenefitsIntegrationOneTimeInfo</td>
<td>0..n</td>
<td>No</td>
</tr>
<tr>
<td>BenefitsIntegrationRecurringInfo</td>
<td>0..n</td>
<td>Yes</td>
</tr>
<tr>
<td>EmpCostAssignment</td>
<td>0..n</td>
<td>Yes</td>
</tr>
<tr>
<td>direct_deposit</td>
<td>0..n</td>
<td>No</td>
</tr>
<tr>
<td>national_id_card</td>
<td>0..n</td>
<td>No</td>
</tr>
<tr>
<td>dependent_information</td>
<td>0..n</td>
<td>Yes</td>
</tr>
<tr>
<td>dependent_relation_information</td>
<td>0..n</td>
<td>Yes</td>
</tr>
<tr>
<td>dependent_personal_information</td>
<td>0..n</td>
<td>Yes</td>
</tr>
<tr>
<td>personal_information{country_code}**</td>
<td>0..1*</td>
<td>Yes</td>
</tr>
<tr>
<td>dependent_address_information</td>
<td>0..n</td>
<td>Yes</td>
</tr>
<tr>
<td>dependent_national_id_card_information</td>
<td>0..n</td>
<td>No</td>
</tr>
<tr>
<td>personal_documents_information</td>
<td>0..n</td>
<td>No</td>
</tr>
<tr>
<td>Segments and Subsegments</td>
<td>Cardinality</td>
<td>Effective-Dated</td>
</tr>
<tr>
<td>-----------------------------------------------</td>
<td>-------------</td>
<td>-----------------</td>
</tr>
<tr>
<td>EmployeeDataReplicationElement</td>
<td>0..n</td>
<td>No</td>
</tr>
<tr>
<td>SecondaryAssignments</td>
<td>0..n</td>
<td>Yes</td>
</tr>
<tr>
<td>SecondaryAssignmentsItem</td>
<td>0..n</td>
<td>Yes</td>
</tr>
<tr>
<td>emergency_contact_primary</td>
<td>0..n</td>
<td>No</td>
</tr>
<tr>
<td>DRTMPurgeStatusOverview</td>
<td>0..n</td>
<td>No</td>
</tr>
<tr>
<td>DRTMPurgeStatus</td>
<td>0..n</td>
<td>No</td>
</tr>
</tbody>
</table>

* There’s at most one segment for each country/region under a `personal_information` segment.

** Time slices of this segment are in sync with time slices of the superordinate segment. This means that there’s only one record of this segment under each superordinate segment.

*** Effective start date and effective end date are taken over from the original parent segment `SecondaryAssignments`.

**More Info and Examples**

The `CompoundEmployee` API supports full transmission (that is, it returns all data as it is on the database with all information available). The API supports the return of delta messages (that is, changed elements only or any information about the action taken). The API returns all employee data according to the selection options and filters. However, only the data of an active employee that has successfully passed through the approval process is returned.

Fields with a null value aren’t returned by the query response (that is, they aren’t added to the response XML document). This rule also applies if the external code of a picklist-based field has a null value.

The `CompoundEmployee` API returns the response field `paycompensation_recurring.annualizationFactor` with value zero if the payment frequency is set to `hourly`. It’s left up to the consumer of the API to calculate the annualization factor.

The query also returns an execution timestamp based on the Employee Central server time when the query was started. On the basis of this time stamp, the next query execution can be triggered using the `last_modified_on` where parameter. Note that the milliseconds part of the timestamp is set to zero to be consistent with the last modified timestamps that have zero milliseconds as well.

**Sample Code**

```xml
<sfobject>
  <id>1</id>
  <type>CompoundEmployee</type>
  <person>
    ...
  </person>
  <execution_timestamp>2013-10-30T14:33:57.000Z</execution_timestamp>
</sfobject>
```

For an example XML, refer to the query response examples in the appendix.
### 2.8.3.2 Employee Completeness Check

Use the employee completeness check to filter out employees whose master data isn’t complete in the CompoundEmployee API query response.

Employee data may not be complete, for example, when a master data import into Employee Central from another system was incomplete.

> **Tip**

Don't do data extractions and data changes on the UI or through imports at the same time, to avoid that incomplete employee data is replicated in the first place.

You have two options to use the completeness check:

- In Provisioning, go to `SFAPI Feature Settings > SFAPI CompoundEmployee`. Define here which segments you want to have included in the employee completeness check:
  - Personal Information
  - Job Information
  - Compensation Information

  If you use this option, the check is performed for that specific company for all processes.

> **Remember**

As a customer, you don’t have access to Provisioning. To complete tasks in Provisioning, contact your implementation partner. If you’re no longer working with an implementation partner, contact Product Support.

- Use query parameter `segmentsForCompleteness` for a process-specific check.

  An error is raised for employees whose data in the following segments is incomplete:
  - `personal_information`
  - `job_information`
  - `compensation_information`

  Use " " or " " to overrule the Provisioning settings and not raise errors for employees with incomplete master data.

> **Restriction**

You can’t apply the completeness check to:

- Employee dependents
- Associated employee information
- Employments where the `hiringNotCompleted` property is set to true
- Compensation information for contingent workers
2.8.3.3 Performance-Improved XML Rendering

The CompoundEmployee API uses the performance-improved XML rendering.

The following features are available with performance-improved XML rendering:

- Dependents, recurring deductions, one-time deductions, alternative cost distribution, and IT declaration are supported by delta transmission.
- The personal_documents_information segment is available.
- Multiple changes for each day to compensation_information are enabled.
- Additional sorting features:
  - Sequence numbers for effective-dated objects that support multiple events for each day are considered when sorting.
  - Semantic keys of effective-dated objects (for example, address information) are considered when sorting.
  - Non-effective-dated objects are sorted according to their semantic keys.
- Performance-optimized rendering and determination of external codes for picklists, foundation objects, and generic object.
- last_modified_on selection by performance optimized SQL.

**i Note**
The performance-improved XML rendering is aligned with OData and SFAPI behavior, therefore behaves as follows:

- Numeric fields with value 0 (null) are included in the result.
- Numeric values are returned as stored in the database, rounding according to the data model isn’t considered.
- Boolean values are returned as false or true.
- External codes for picklist values are solely determined based on the picklist option ID. Therefore, if the picklist assigned to a field in the data model is changed, the API keeps returning the former external code until the picklist values are updated, for example, using the UI.

If you experience incompatibilities with the performance-improved XML rendering, you can switch to the compatibility mode.

**Related Information**

Compatibility Mode [page 43]
Personal Documents [page 57]
2.8.3.4 Compatibility Mode

If you use the performance-improved XML rendering and experience incompatibilities, you can enable the compatibility mode for the CompoundEmployee API.

Enable the compatibility mode in Provisioning under API Settings and Tools > SFAPI Feature Settings > Enable compatibility mode (use only in alignment with SuccessFactors).

⚠️ Caution
Enable the compatibility mode only in exceptional cases since it increases the API execution time. Please contact Product Support before doing so.

→ Remember
As a customer, you don’t have access to Provisioning. To complete tasks in Provisioning, contact your implementation partner. If you’re no longer working with an implementation partner, contact Product Support.

The compatibility mode behaves as follows:

- On the jobInfo segment, it ignores the following fields if their value is 0:
  - FTE
  - SHIFT_FACTOR
  - STANDARD_HOURS
  - WORKING_DAYS_PER_WEEK
  - AMOUNT_OF_FINANCIAL_PLAN
  - ERN_NUMBER

- On the compInfo segment, it ignores the following fields if their value is 0:
  - BENEFITS_RATE
  - PAY_MONTHS_PER_YEAR
  - SALARY_RATE_UNITS
  - RANGE_PENETRATION

- On the PayCompensationRecurring and PayCompensationNonRecurring segments:
  - It rounds Value (=amount) and CalculatedAmount
  - It rounds NumberOfUnits
  - It ignores CalculatedAmount if it is 0

Related Information

Performance-Improved XML Rendering [page 42]
2.8.3.5 Dynamic Field List

The CompoundEmployee API returns a hybrid field list contained in the query response. The dynamic field list consists of a static set of fields and all fields that are set to visibility edit, view, or both in any data model. Fields that are both in the static set of fields and visible in any of the data models are only included once in the hybrid field list to avoid duplicates. Even if no fields are active in the data model, the fields of the static set of fields are always returned.

For HRIS-based objects, fields that aren’t visible in any data model and aren’t included in the static set of fields either are not returned by the API, even if there’s data stored in the database.

For MDF-based objects, all business-relevant elements that are modeled in the object definition are returned in the response, independent from the visibility defined in the object definition and whether there’s data in the database or not. System-administrative fields like, for example, mdfSystemLastModifiedBy or mdfSystemCreatedDate are returned as well for MDF objects. Other system fields (mdfSystem…) are omitted from the CompoundEmployee API response.

If data is stored in the database, country/region-specific fields are always included in the query response, no matter if they’re part of the static or the dynamic field list.

Transient fields are configurable, however, they’re ignored by the CompoundEmployee API.

Related Information

Transient Fields [page 163]

2.8.3.6 Picklists, Foundation Objects, and Generic Objects

See how the CompoundEmployee API handles picklists, foundation objects, and generic objects.

The CompoundEmployee API returns the external codes for fields based on foundation objects or generic objects. For fields based on picklists it returns non-unique external codes. If the external code or non-unique external code can’t be determined, the field is omitted and a log item is added to the response.

For time-dependent foundation objects, time-dependent determination of external codes isn’t supported. The external code of the last time slice is returned. Keep the external code stable for the lifetime of the object. If you change the external code, you must change it in all time slices of the foundation object.

For country/region-specific picklists, the external code is determined from the picklist maintained for the relevant field in the Country/Region-Specific Succession Data Model.

Fields for which corresponding non-unique external code respectively external codes aren’t available or aren’t maintained in the picklist, foundation table or generic object aren’t returned by the query response.

Field entries with invalid references (for example incorrect picklist option ID) are returned by the query response. But the field in question is left empty and a log item is added to the person segment explaining the issue.
If the value of a field refers to a picklist, generic object instance, or foundation object entry marked as deleted, obsolete, or inactive, the fields are processed by the CompoundEmployee API and corresponding non-unique external codes or external codes are returned with the query response (regardless of the status of the picklist, generic object instance, or foundation table entry). Physically deleted picklist entries and foundation table entries aren’t returned by the query response. To avoid confusion, make sure that the non-unique external codes of your picklists are unique in the context of the picklist ID.

### 2.8.3.7 Date/Time Handling

CompoundEmployee API query expressions handle date and time in different ways, depending on whether the `last_modified_on` or the `effective_end_date` parameter is used.

Query expressions using the `last_modified_on` parameter consider the time zone specified in the query expression as well as the time zone stored in the database records. Last modified dates are always returned in a Universal Time Coordinated (UTC) timestamp.

**Example**

Let’s say, the last modified date of Personal Information on the database is `'2012-11-07T00:00:00+01:00'` (Berlin time). Then the query expression `last_modified_on >= '2012-11-06T23:30:00Z'` (UTC) doesn’t return this Personal Information record because `'2012-11-07T00:00:00+01:00'` in Berlin time is `'2012-11-06T23:00:00Z'` in UTC and thus is earlier than the time specified in the query expression.

With respect to time zones, the handling for the `effective_end_date` parameter in query expressions differs from the handling of the `last_modified_on` parameter. For `effective_end_date`, the API doesn’t take the time zone into account. The date specified in the query is taken as is. It is compared to the date that’s part of the effective end date stored in the database, without converting both or one of these dates to UTC. This can lead to unexpected behavior and requires the application consuming the CompoundEmployee API to consider the system time zone of the Employee Central system.

For effective-dated tables that have a sequence counter, only the record with the highest sequence counter is returned if multiple records exist for the same time slice. This affects the subsegments of the `job_information`, `compensation_information`, and `paycompensation_recurring` segments.

**Example**

Let’s say, the effective start date of Job Information on the database is `'2012-11-07T00:00:00+01:00'` (Berlin time). In UTC, this equals `'2012-11-06T23:00:00Z'`. Then a query expression `effective_end_date >= '2012-11-07'` returns this Job Information record because `'2012-11-07T00:00:00'` (query string) is later than or equals `'2012-11-07T00:00:00'` (database). The query string and the database date/time haven’t been converted to UTC.

The `created_on_timestamp` field allows API consumers to distinguish between the hire of an employee and a subsequent change of the first job time slice. When the `last_modified_on` value is later than the `created_on_timestamp` of the job record, the job record has been corrected after the original hire of the employee and the consumer may have already processed the hire.
2.8.3.8 Logon User ID and User Characteristics

The CompoundEmployee API query response holds information about the logon user.

The logon user ID is published as a calculated element in the segment person with field name logon_user_id.

If employment information isn’t requested by the query select statement, the logon user ID isn’t populated in the query response. The same applies to the fields logon_user_name and logon_user_is_active.

If Concurrent Employment is enabled, logon information can only be partly determined. The fields logon_user_name and logon_user_is_active are derived from columns ACCOUNT_STATUS and USERNAME, respectively, of object USER_ACCOUNT. In this case, the field logon_user_id is omitted.

In general, the logon user ID can only be returned if the employee has only one employment. Whenever an employee has multiple employments, the logon user ID is omitted. The exceptions to this rule are Global Assignment and Higher Duty. If an employee has one standard employment and one or more employments for Global Assignment or Higher Duty, the logon user ID of the standard employment is returned.

If you include the assignment_class filter in the query request and use other filter values than ST (that is, standard employment), the query response doesn’t populate the fields logon_user_id, logon_user_name, and logon_user_is_active for the employee.

Related Information

assignment_class Employment Filter [page 58]

2.8.3.9 Employee Central Entity Semantic Keys

Some Employee Central entities have semantic keys that are used in the CompoundEmployee API query response to identify and distinguish records of said entities and include the parent segment ID.

<table>
<thead>
<tr>
<th>Parent Segment</th>
<th>Employee Central Entity</th>
<th>Semantic Key</th>
</tr>
</thead>
<tbody>
<tr>
<td>person</td>
<td>address_information</td>
<td>address_type, start_date, and country</td>
</tr>
<tr>
<td>person</td>
<td>email_information</td>
<td>email_type</td>
</tr>
<tr>
<td>person</td>
<td>direct_deposit</td>
<td>deposit_type, routing_number, account_number, account_type, and process_type</td>
</tr>
<tr>
<td>person</td>
<td>national_id_card</td>
<td>country and card_type</td>
</tr>
<tr>
<td>person</td>
<td>personal_information_global</td>
<td>start_date and country</td>
</tr>
</tbody>
</table>
2.8.3.10 **Country/Region-Specific Fields**

For the country/region-specific fields in the segments job information, compensation information, and employment information external code values of picklists, foundation objects, and generic object instances are returned by the CompoundEmployee API. No matter whether the model is active or not, as long as there’s data in the database for those specific fields.

**Country/Region-Specific Fields in Personal Information** [page 47]

The CompoundEmployee API returns country/region-specific personal information in the query response.

**Country/Region-Specific Fields in Address Information** [page 48]

Addresses for non-localized countries/regions are displayed in the CompoundEmployee API query response.

2.8.3.10.1 **Country/Region-Specific Fields in Personal Information**

The CompoundEmployee API returns country/region-specific personal information in the query response.

Country/region-specific personal information is returned by the API as subsegments personal_information_<country> of segment personal_information. For every personal information record either none, one, or more country/region-specific personal information records exist (at most only one per country/region).

**Query Request**

Country/region-specific personal information segments aren’t supported as parameters in the SELECT statement, but are automatically returned if the personal_information node is part of the SELECT clause.
In the `WHERE` expression, the table with country/region-specific personal information is considered for queries with the field `last_modified_on` as well.

**Query Response**

The query response structure is as follows:

```
<queryResponse>
  <result>
    <sfobject>
      <id>112</id>
      <type>CompoundEmployee</type>
      <person>
        ...
        <personal_information>
          <start_date>...</start_date>
          <end_date>...</end_date>
          ...
          <personal_information_{country/region1}>
        </personal_information_{country/region1}>
        ...
        <personal_information_{country/region2}>
        ...
        </personal_information>
      ...
    </sfobject>
  ...
</result>
</queryResponse>
```

The XML contains the tags `<personal_information_{country/region}>`, which carry the country/region-specific information. The placeholders `{country/region1}` and `{country/region2}` represent the territory codes.

**Note**

The segments `personal_information_{country/region}` contains no fields for `start_date` and `end_date`, as both are already given by the parent segment `personal_information`.

### 2.8.3.10.2 Country/Region-Specific Fields in Address Information

Addresses for non-localized countries/regions are displayed in the `CompoundEmployee API` query response.

The element `<is_global_modal_address>` indicates if an address is modeled in the global (meaning not country/region-specific) Succession Data Model or in the Country/Region-Specific Succession Data Model. If the address is modeled in the global Succession Data Model, the value `true` is returned for the element `<is_global_modal_address>`. If the address is modeled in the Country/Region-Specific Succession Data Model, the value `false` is returned.
2.8.3.11 External Key Mapping for Cost Centers

The CompoundEmployee API supports external key mapping for cost centers. If you decide to use external key mapping for cost centers, you need to maintain the field *External Object ID* of the Cost Center MDF object. If this external code is to be returned in the API response, the `externalKeyMapping` parameter must be specified in the query as follows:

```xml
<urn:param>
  <urn:name>externalKeyMapping</urn:name>
  <urn:value>costCenter</urn:value>
</urn:param>
```

In the CompoundEmployee API, the value of this external code field is then used instead of the Employee Central cost center code. This applies to the cost center key in the following segments:

- `job_information`
- `alternative_cost_distribution_item`
- `paycompensation_non_recurring`
- `EmpCostAssignment`

2.8.3.12 Global Assignments

The CompoundEmployee API returns global assignment information in the query response. The `global_assignment_information` segment resides in the `employment_information` segment. There can be at most one `global_assignment_information` segment beneath an `employment_information` segment.

Most fields of the `global_assignment_information` segment are also contained in the `employment_information` segment, except the custom fields `customString101` through `customString120`. These fields aren't contained in the model of the `employment_information` segment.

Each employment record has a unique `USERS_SYS_ID`. For global assignments (host employments), separate employments are created so that the `USERS_SYS_ID` remains unique for each employment.

2.8.3.13 Secondary Assignments

The CompoundEmployee API query response can also contain information about secondary assignments.

**Note**

Secondary assignment information can only be returned if *Concurrent Employment Management* is enabled in SAP SuccessFactors.

The `SecondaryAssignments` segment is a subelement under the `person` segment of the API response. The subitem `SecondaryAssignmentsItem` is additionally replicated as `SecondaryAssignmentPeriod` under the corresponding `employment_information` segment.
To have the API extract secondary assignments, add the select item `SecondaryAssignments`. This MDF object is effective dated. Each instance of such an object contains a list of items that denote those employments, which are secondary assignments in the time period of that object instance.

### 2.8.3.14 Payment Information

The `CompoundEmployee` API returns MDF-based Payment Information data in the query response.

**Prerequisites for Using MDF-Based Payment Information**

The MDF-based Payment Information must be enabled in Provisioning.

- **i Note**
  
  Once you’ve enabled the MDF-based Payment Information, you can’t go back to the HRIS-based Direct Deposit/Payment Information.

  For more information, refer to the [Employee Central Payment Information](https://help.sap.com) implementation guide in SAP Help Portal.

**Subsegment for MDF-Based Payment Information**

The MDF-based Payment Information is an effective-dated subsegment of the `employment_information` segment and has the following structure:

```xml
<queryResponse>
  <result>
    <sfobject>
      <id>112</id>
      <type>CompoundEmployee</type>
      <person>
        ...
        <employment_information>
          ...
          <PaymentInformationV3>
            <PaymentInformationDetailV3>
              ...
              <PaymentInformationDetailV3"country">
                ...
              </PaymentInformationDetailV3
            </PaymentInformationV3
          </employment_information>
        </person>
      ...
    </sfobject>
  </result>
</queryResponse>
```

Along with the MDF-based Payment Information comes a change history feature for the detection of changes and deletions. You need to enable this feature in the object definition of each MDF object. If it isn’t switched on, the API errors out and terminates the processing of the query.
Compositions to custom objects are considered by this approach and are returned by the CompoundEmployee API response.

### 2.8.3.15 Person Relationship and Employee Dependents

The CompoundEmployee API can return the employees’ dependents and also their person relationships in the query response.

#### Employee Dependents

Employee dependents are persons related to an employee without actually being employed by the company, for example, the spouse of an employee. These relationships between employees are returned in the person_relation segment and their dependent data is returned in the segment dependent_information.

Information about the dependent is queried by the API using the select item dependent_information. All information belonging to the dependent is available as child segments. This means that all time slices for the effective-dated objects are returned under the dependent_information segment. The child segments are:

- dependent_address_information
- dependent_personal_information (including country/region-specific information)
- dependent_national_id_card_information
- dependent_relation_information

Dependent information can be selected independently of the person relation information. The relationship information is part of the child segment dependent_relation_information. The only difference is that PERSON_RELATION_ID isn’t included, since it’s already included in the dependent information segment.

The query result structure for dependent information looks as follows:

```xml
<person>
  ...
  <dependent_information>
    ...
    <person_id>123</person_id>
    <person_id_external>112_d123</person_id_external>
    ...
    <dependent_relation_information>
      ...
      <end_date>9999-12-31</end_date>
      <is_accompanying_dependent>true</is_accompanying_dependent>
      ...
      <relationship_type>1</relationship_type>
      <start_date>2015-01-01</start_date>
    </dependent_relation_information>
    <dependent_address_information>
      ...
      <start_date>2013-03-08</start_date>
      <zip_code>123456</zip_code>
    </dependent_address_information>
  </dependent_information>
</person>
```
**Person Relationship**

person_relation is a subelement under the person segment in the CompoundEmployee API's response. It's effective dated.

The query result structure for Person Relationship information looks as follows:

```xml
<queryResponse>
  <result>
    <sfobject>
      <id>112</id>
      <type>CompoundEmployee</type>
      <person>
        ...
        <person_relation>
          <start_date>...</start_date>
          <end_date>...</end_date>
          <person_id>123</person_id>
          <dependency_type>...</dependency_type>
          ...
        </person_relation>
        <employment_information>
          ...
          <accompanying_dependent>
            ...
            </accompanying_dependent>
        </employment_information>
      </person>
      ...
    </sfobject>
  </result>
</queryResponse>
```

**i Note**

The subelement accompanying_dependent is deprecated and only supported for compatibility reasons as a select item.
2.8.3.16 Emergency Contact

Enable the emergency_contact_primary segment in the CompoundEmployee API query response by adding the emergency_contact_primary select item to the select statement.

The emergency_contact_primary segment is rendered underneath the person segment and lists information about an employee’s emergency contacts.

**Note**

If the business keys of multiple emergency contacts are identical, deltas can’t be calculated for emergency_contact_primary when using delta transmission. If so, the person as a whole isn’t rendered and an error message is returned. The semantic key is made up of the keys name and relationship.

**Restriction**

Currently only basic emergency contact information is considered. Address information for emergency contacts isn’t returned by the API.

Related Information

SELECT Clause [page 21]

2.8.3.17 Associated Employee Information

To get associated employee information, specify the selection items employment_information and associated_employee_information along with job_information or job_relation in the query.

Associated employee information is employment information or job information of the employee’s supervisor or other job-related employees. If you specify neither job_information nor job_relation in the query, the select item associated_employee_information is ignored by the API.

In the query response, the associated employee information is displayed as a subsegment of employment_information. The associated_employee_information segment itself holds the person information and has the segments associated_employee_employment and associated_employee_job_information associated to it.

**Remember**

Since reading and rendering associated employee information is very time consuming, only use it in processes where this specific information is needed.
2.8.3.18 One-Time and Recurring Information for Benefits Integration

Find out about the benefits integration segments of the CompoundEmployee API, which can be used to transfer amounts such as insurance premiums or contributions to savings plans to a payroll system.

Benefits Information in Employee Central

Some benefits amounts aren’t a payment or a deduction. For example, insurance premiums or contributions to savings plans. Employee Central stores this type of amounts in two different objects so that they can be transferred to the payroll system:

- **One-Time Information for Benefit Integration** (BenefitsIntegrationOneTimeInfo)
  This object isn’t effective-dated.
- **Recurring Information for Benefit Integration** (BenefitsIntegrationRecurringInfo)
  This object is effective-dated.

Benefits Information in the CompoundEmployee API

The CompoundEmployee API has segments for the benefits information objects, which are included as child segments in employment_information:

- BenefitsIntegrationOneTimeInfo
  This segment has only one date field, payCompDate, which defines when the benefit is due.
- BenefitsIntegrationRecurringInfo
  This segment has the effective-dated fields payCompBeginDate and payCompEndDate.

The segments have the same structure. They don’t have any child segments.

You can request each segment separately.

Benefits Information in the Query Response Structure

Here’s what the query response looks like if you add the BenefitsIntegrationRecurringInfo segment to the API request:

```
<queryResponse>
  <result>
    <sfobject>
      <id>112</id>
      <type>CompoundEmployee</type>
      <person>
        ...
    </person>
  </result>
</queryResponse>
```
2.8.3.19 Employee Cost Assignment

Find out about the Employee Cost Assignment segment of the CompoundEmployee API, which can be used to replicate business information like funds and grants to a payroll system.

Employee Cost Assignment in the Query Response Structure

The CompoundEmployee API supports the employee cost assignment object, which is included as child segment in employment_information. The EmpCostAssignmentItem child segment contains all business details.

i Note

This object supports external key mapping with the externalKeyMapping parameter and the costCenter value. For more information, see External Key Mapping for Cost Centers.

Here's what the query response looks like if you add the EmpCostAssignment segment to the API request:

```xml
<queryResponse>
  <result>
    <sfobject>
      <id>920</id>
      <type>CompoundEmployee</type>
      <person>
        ...<employment_information>
          ...
          <BenefitsIntegrationRecurringInfo>
            <benefit>1232</benefit>
            <benefitEnrollment>58E706A3578F451D94EFD28E7203D868</benefitEnrollment>
            <benefitSchedulePeriod>2342</benefitSchedulePeriod>
            <createdBy>root</createdBy>
            <createdDate>2019-12-03T07:00:08.000Z</createdDate>
            <currency>USD</currency>
            <externalName>hwilson1_HealthInsurance_INSURANCE_EMPLOYER_CONTRIBUTION</externalName>
            <frequency>ANN</frequency>
            <id>0AFB4C5DFECD40469149CBD1ACF6C6D8</id>
            <lastModifiedBy>root</lastModifiedBy>
            <lastModifiedDate>2019-12-03T07:00:08.000Z</lastModifiedDate>
            <payCompBeginDate>2021-01-01</payCompBeginDate>
            <payCompEndDate>9999-12-31</payCompEndDate>
            <payComponent>6454</payComponent>
            <value>50.0</value>
          </BenefitsIntegrationRecurringInfo>
          ...
        </person>
      </sfobject>
    </result>
  </queryResponse>
```
2.8.3.20 Income Tax Declaration

The Income Tax Declaration object is supported in the CompoundEmployee API in full transmission mode and can be enabled by including the select parameter ItDeclaration in the select items.

The ItDeclaration segment is under the employment segment. The query response structure for this segment looks like this:

```xml
<queryResponse>
  <result>
    <sobject id="112">
      <type>CompoundEmployee</type>
      <person>...
        <employment_information>...
          <ItDeclaration>
            <amount>250.0</amount>
            <approvalStatus>PENDING</approvalStatus>
            <createdBy>TDenman</createdBy>
            <createdDate>2014-12-10T10:08:15.000Z</createdDate>
            <declarationType>ACTUAL</declarationType>
            <effectiveEndDate>9999-12-31</effectiveEndDate>
            <effectiveStartDate>2014-12-10</effectiveStartDate>
            <fiscalYear>2014</fiscalYear>
            <itDeclInvestmentType>IV1_code</itDeclInvestmentType>
            <lastModifiedBy>TDenman</lastModifiedBy>
            <lastModifiedDate>2014-12-10T10:08:15.000Z</lastModifiedDate>
            <postingDate>2014-12-10</postingDate>
            <type>D2</type>
          </ItDeclaration>
        </employment_information>
      </person>
    </sobject>
  </result>
</queryResponse>
```
2.8.3.21 Personal Documents

The CompoundEmployee API query returns information about employee documents, but not the document itself.

The `personal_documents_information` segment lists information such as document title, document type, and place and country/region of issue.

⚠️ Caution

The `personal_documents_information` segment is only available if the performance-improved XML rendering is enabled.

Enable `personal_documents_information` by adding the SELECT item `personal_documents_information` to the SELECT statement. The segment is rendered under the `employment_information` segment.

Related Information

Performance-Improved XML Rendering [page 42]

2.8.3.22 Contingent Worker Employment Filter

The CompoundEmployee API can filter contingent workforce employments and regular employments of an employee.

To define whether contingent workforce employments or regular employments are filtered, use the `isContingentWorker` select parameter. If there are employees in Employee Central that only have one type of employment, the selection of the whole person depends on the select parameter.

It's possible that one employee has both types of employment. In this case, the selection parameter `isContingentWorker` filters out the employment type that wasn't requested.

<table>
<thead>
<tr>
<th>Condition in Query Request</th>
<th>Regular Employment</th>
<th>Contingent Worker Employment</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>isContingentWorker = false</code> (default)</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td><code>isContingentWorker = true</code></td>
<td>No</td>
<td>Yes</td>
</tr>
</tbody>
</table>
2.8.3.23 hiringNotCompleted Employment Filter

Use the hiringNotCompleted parameter in the query request of the CompoundEmployee API if you want your downstream systems to receive only data that has been validated against the employee data model in Employee Central.

The hiringNotCompleted parameter evaluates the indicator property hiringNotCompleted in the EmpEmployment entity in Employee Central, which was introduced for Onboarding. This property allows for differentiating data records of candidates (that is, new hires that didn’t yet complete the Manage Pending Hire process). For more information about Onboarding, refer to Implementing Onboarding.

The hiringNotCompleted field is part of the employment_information segment. If the value for hiringNotCompleted of an employment is false, it’s a standard employment of a hired employee. If the value for hiringNotCompleted of an employment is true, it’s the employment of a candidate. That is, the employment is possibly incomplete with respect to the employee data model of Employee Central.

hiringNotCompleted = true isn’t supported by the API. This means that you can’t use the hiringNotCompleted filter to only return employments of candidates. Returning employments of candidates is only possible if you don’t use hiringNotCompleted at all in the WHERE condition. In this case, standard employments of hired employees and also (possibly) incomplete employments of candidates are contained in the query response.

The hiringNotCompleted filter is used by standard integrations to prevent the replication of incomplete employments.

<table>
<thead>
<tr>
<th>Query Request</th>
<th>Query Response</th>
</tr>
</thead>
<tbody>
<tr>
<td>Contains hiringNotCompleted = false</td>
<td>Contains employments of hired employees</td>
</tr>
<tr>
<td></td>
<td>Doesn’t contain employments of candidates</td>
</tr>
<tr>
<td>Doesn’t contain hiringNotCompleted</td>
<td>Contains employments of hired employees as well as candidates</td>
</tr>
</tbody>
</table>

2.8.3.24 assignment_class Employment Filter

Use the assignment_class parameter in the query request if you want to filter employments for their assignment type such as standard employment or global assignment.

You can use the assignment_class parameter to filter employments by the ASSIGNMENT_TYPE field from the employment in Employee Central. Possible filter values are, for example, ST for standard employments or...
GA for global assignments. If you don't include ST in the filter values, the query response doesn't contain logon_user_id, logon_user_name, and logon_user_is_active for the employee.

The filter is case-sensitive.

Filtering for assignment_class is only supported in the standard query mode.

Table 2: Use of the assignment_class Employment Filter

<table>
<thead>
<tr>
<th>Query Request</th>
<th>Query Response</th>
</tr>
</thead>
<tbody>
<tr>
<td>Contains assignment_class = 'ST'</td>
<td>Contains employments with the assignment type Standard Employment</td>
</tr>
<tr>
<td></td>
<td>Doesn't contain employments with other assignment types, such as Global Assignment</td>
</tr>
<tr>
<td>Contains assignment_class = 'GA'</td>
<td>Contains employments with the assignment type Global Assignment</td>
</tr>
<tr>
<td></td>
<td>Doesn't contain employments with other assignment types, such as Standard Employment</td>
</tr>
<tr>
<td>Contains assignment_class in ('ST', 'GA')</td>
<td>Contains employments with the assignment types Standard Employment or Global Assignment</td>
</tr>
<tr>
<td></td>
<td>Doesn't contain employments with other assignment types</td>
</tr>
<tr>
<td>Doesn't contain assignment_class</td>
<td>Contains all employments regardless of their assignment type</td>
</tr>
</tbody>
</table>

Related Information

Logon User ID and User Characteristics [page 46]

2.8.3.25 Start Date Condition

Use the START_DATE parameter of the CompoundEmployee API when you want to receive employee data because the employee has time slices starting after a specific date.

When the START_DATE select parameter is used, the API returns all employees for which at least one of the requested segments becomes effective on or after the requested date. A segment is effective when it has an effective_start_date that is later than or equal to the date provided by the consumer. The parameter is only relevant for effective-dated segments. Non-effective-dated segments don't have a specific date marking the start of the data record. The only exception from this rule is the segment paycompensation_non_recurring, which uses the pay date instead of the start date. The START_DATE condition can't be used for MDF objects (payment information, cost distribution, deductions).

Because the START_DATE condition is applied to each of the requested segments, the employee is also returned if the condition is only true for one of the requested segments.
Restriction

The START_DATE parameter is only supported in full transmission mode and not in delta or snapshot mode.

The START_DATE condition can’t be used in combination with the LAST_MODIFIED_ON condition and the SNAPSHOT_DATE condition.

Example

Here’s an example of how the start date is handled. Let’s say, the employee Paul was created (hired) during the last payroll period (payroll period 0), but isn’t effective until the current payroll period (payroll period 1). Jane is created (hired) in the current payroll period and is also effective in the current payroll period.

If the LAST_MODIFIED_ON condition was used in the API request, Paul has already been replicated with the last replication. However, the consumer might have filtered out Paul’s data because it wasn’t relevant at that point in time. If you use the START_DATE condition in the next replication run, Paul is replicated again because he’s now effective in this payroll period.

Figure 2: Using the START_DATE Condition

Segments Supported by the Start Date Condition [page 61]

Find out which segments are supported or not supported by the START_DATE condition of the CompoundEmployee API.

Differences Between Start Date, Effective End Date, and Last Modified Conditions [page 62]

The CompoundEmployee API query handles the start date, end date, and last modified date in different ways.

Particularities and Data Constellations to Be Aware Of [page 64]

Be aware of the following issues when using the start_date condition of the CompoundEmployee API.

What to Consider When Filtering the Response for the Start Date [page 68]

If you want to filter the employee information you get from the CompoundEmployee API when applying the START_DATE condition, you must be aware of the difficulties that can occur when filtering data.
## 2.8.3.25.1 Segments Supported by the Start Date Condition

Find out which segments are supported or not supported by the `START_DATE` condition of the `CompoundEmployee API`.

<table>
<thead>
<tr>
<th>Segment / Segments</th>
<th>Supported</th>
<th>Field Relevant for the Condition</th>
<th>Supported Only in Combination with Supported Segments</th>
<th>Why Not Supported</th>
</tr>
</thead>
<tbody>
<tr>
<td>person</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>personal_information</td>
<td>Yes</td>
<td>effective_start</td>
<td>No</td>
<td></td>
</tr>
<tr>
<td><code>Personal_information_(country_code)</code></td>
<td>Yes</td>
<td>effective_start</td>
<td>No</td>
<td></td>
</tr>
<tr>
<td>address_information</td>
<td>Yes</td>
<td>effective_start</td>
<td>No</td>
<td></td>
</tr>
<tr>
<td>email_information</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>phone_information</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>person_relation</td>
<td>Yes</td>
<td>effective_start</td>
<td>No</td>
<td></td>
</tr>
<tr>
<td>employment_information</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>global_assignment_information</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>job_information</td>
<td>Yes</td>
<td>effective_start</td>
<td>No</td>
<td></td>
</tr>
<tr>
<td>alternative_cost_distribution</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>MDF Object</td>
</tr>
<tr>
<td>alternative_cost_distribution_item</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>MDF Object</td>
</tr>
<tr>
<td>compensation_information</td>
<td>Yes</td>
<td>effective_start</td>
<td>No</td>
<td></td>
</tr>
<tr>
<td>paycompensation_recurring</td>
<td>Yes</td>
<td>effective_start</td>
<td>No</td>
<td></td>
</tr>
<tr>
<td>paycompensation_non_recurring</td>
<td>Yes</td>
<td>pay_date</td>
<td>No</td>
<td></td>
</tr>
<tr>
<td>payment_information</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>MDF Object</td>
</tr>
<tr>
<td>accompanying_dependent</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td></td>
</tr>
<tr>
<td>job_relation</td>
<td>Yes</td>
<td>effective_start</td>
<td>No</td>
<td></td>
</tr>
<tr>
<td>deduction_recurring</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>MDF Object</td>
</tr>
<tr>
<td>deduction_recurring_item</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>MDF Object</td>
</tr>
</tbody>
</table>
### Segment / Segments

<table>
<thead>
<tr>
<th>Segment / Segments</th>
<th>Supported</th>
<th>Field Relevant for the Condition</th>
<th>Supported Only in Combination with Supported Segments</th>
<th>Why Not Supported</th>
</tr>
</thead>
<tbody>
<tr>
<td>deduction_non_recurring</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>MDF Object</td>
</tr>
<tr>
<td>ItDeclaration</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>MDF Object</td>
</tr>
<tr>
<td>PaymentInformationV3</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>MDF Object</td>
</tr>
<tr>
<td>associated_employee_information</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td></td>
</tr>
<tr>
<td>direct_deposit</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>national_id_card</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>dependent_information</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td></td>
</tr>
<tr>
<td>SecondaryAssignments</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>MDF Object</td>
</tr>
<tr>
<td>SecondaryAssignmentsItem</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>MDF Object</td>
</tr>
<tr>
<td>emergency_contact_primary</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td></td>
</tr>
</tbody>
</table>

### 2.8.3.25.2 Differences Between Start Date, Effective End Date, and Last Modified Conditions

The CompoundEmployee API query handles the start date, end date, and last modified date in different ways.

#### Start Date and End Date Conditions

If employees are replicated due to the `start_date` condition, they're always sent as a whole with all data records. If the consumer wants to filter this data additionally, they can filter the API response. Alternatively, the `effective_end_date` condition can be used in combination with the `start_date` condition. When using these two conditions, be aware of their differences.

<table>
<thead>
<tr>
<th>Start Date Condition</th>
<th>Effective End Date Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Influences SQL statement</td>
<td>Is a filter applied on the result of an SQL request</td>
</tr>
<tr>
<td>Looks for time slices that have a start date equal to or later than the provided date</td>
<td>Checks which time slices are effective on the provided date</td>
</tr>
</tbody>
</table>

Both the `start_date` and the `effective_end_date` condition have a common date format: `to_date('2014-06-01', 'YYYY-MM-DD')`

A combination of the `start_date` and the `effective_end_date` condition only makes sense if the same date is used in both conditions. Otherwise, the segment that fulfills the `start_date` condition and leads to the
replication of the employee can be filtered out because the effective_end_date condition isn’t true for this segment.

The following example shows you how the start_date and the effective_end_date condition differ with regard to their results and what happens if both are used in combination.

The initial situation is as follows:

- Job slice 1 begins on January 1, 2014
- Job slice 2 begins on June 15, 2014

In the first case, the current payroll period starts on June 1, 2014.

Table 4: Differences Between Start Date and Effective End Date – Payroll Period Starting on June 1

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Condition</th>
<th>What the Condition Applies To</th>
<th>What’s Returned</th>
</tr>
</thead>
<tbody>
<tr>
<td>start_date</td>
<td>start_date &gt;= to_date('2014-06-01','YYYY-MM-DD')</td>
<td>Job slice 2</td>
<td>All job slices</td>
</tr>
<tr>
<td>effective_end_date</td>
<td>effective_end_date=to_date('2014-03-01','YYYY-MM-DD')</td>
<td>Job slice 1</td>
<td>Job slice 1</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>If &gt;= instead of =: job slice 2 is returned as well</td>
</tr>
<tr>
<td>start_date and</td>
<td>start_date &gt;= to_date('2014-06-01','YYYY-MM-DD') AND</td>
<td>Job slice 2 (due to start date condition)</td>
<td>Job slice 1 (job data is filtered)</td>
</tr>
<tr>
<td>effective_end_date</td>
<td>effective_end_date = to_date('2014-06-01','YYYY-MM-DD')</td>
<td></td>
<td></td>
</tr>
<tr>
<td>start_date and</td>
<td>start_date &gt;= to_date('2014-06-01','YYYY-MM-DD') AND</td>
<td>Job slice 2 (due to start date condition)</td>
<td>All job slices</td>
</tr>
<tr>
<td>effective_end_date</td>
<td>effective_end_date &gt;= to_date('2014-06-01','YYYY-MM-DD')</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

In the second case, the current payroll period starts on July 1, 2014.

Table 5: Differences Between Start Date and Effective End Date – Payroll Period Starting on July 1

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Condition</th>
<th>What the Condition Applies To</th>
<th>What’s Returned</th>
</tr>
</thead>
<tbody>
<tr>
<td>start_date</td>
<td>start_date &gt;= to_date('2014-07-01','YYYY-MM-DD')</td>
<td>Neither job slice</td>
<td>Neither job slice</td>
</tr>
<tr>
<td>effective_end_date</td>
<td>effective_end_date = to_date('2014-07-01','YYYY-MM-DD')</td>
<td>Job slice 2</td>
<td>Job slice 2</td>
</tr>
<tr>
<td>Parameter</td>
<td>Condition</td>
<td>What the Condition Applies To</td>
<td>What's Returned</td>
</tr>
<tr>
<td>-------------------------</td>
<td>---------------------------------------------------------------------------</td>
<td>-------------------------------</td>
<td>-----------------</td>
</tr>
<tr>
<td>start_date and effective_end_date</td>
<td>start_date &gt;= to_date('2014-07-01','YYYY-MM-DD') AND effective_end_date = to_date('2014-07-01','YYYY-MM-DD')</td>
<td>Neither job slice</td>
<td>Neither job slice</td>
</tr>
</tbody>
</table>

### Start Date and Last Modified On Conditions

Like the `start_date` condition, the `last_modified_on` condition influences the SQL statement directly. It returns the employee in case the condition is true for any of the requested segments. By adjusting the provided date in the `last_modified_on` condition to the last replication date, it can be ensured that subsequent calls only lead to employee replication if the employee has changed. In contrast to this, an API call with the `start_date` condition can lead to multiple identical results.

For regular employee replication, it makes sense to use the `last_modified_on` condition. The `start_date` condition is useful in case the consumer can only work with current data and not store future time slices. A typical use case is a payroll system that is only interested in the records of the current payroll period. The payroll system calls the `CompoundEmployee` API with the last modified condition but ignores future time slices. In order to receive these future time slices when they become relevant, the `start_date` condition is used additionally. If the `CompoundEmployee` API is called more than once per payroll period, it's sufficient to use the `last_modified_on` condition in all cases and only additionally use the `start_date` condition for the first API call of the payroll period.

### 2.8.3.25.3 Particularities and Data Constellations to Be Aware Of

Be aware of the following issues when using the `start_date` condition of the `CompoundEmployee` API.

#### Non-Effective-Dated Segments

The `start_date` condition isn’t supported for non-effective-dated segments (exception: `paycompensation_non_recurring`). This means that, for example, if you delete the email information when using the `start_date` condition, no response is returned.

Use other SFQL requests or the `last_modified_on` condition to detect changes and deletions of non-effective-dated segments.
Retrograde Changes

The start_date last_modified_on condition.

Example

Such a change would be if in February, a spot bonus is assigned to an employee (paycompensation_non_recurring) with pay date January 1. Or compensation is increased with January 1 as effective start date. condition can't be used to detect changes on time slices in the past. The reason is that the date provided by the consumer is incremented with every API call. A typical use case is the monthly payroll process. Changes in the past aren’t detected. In this case, use the

Employee Selection Due to Deletions and Shifts

The start_date condition takes historic changes into consideration. Therefore, the CompoundEmployee condition can’t be used to detect changes on time slices in the past. The reason is that the date provided API might return employees in situations that aren’t immediately obvious to the consumer. This happens when time slices are deleted or shifted where the consumer doesn’t see the changes on the UI and can’t track them.

The CompoundEmployee API must return the employee in these cases because it can’t know whether or not the consumer has filtered out the deletion or shift when it was originally communicated by the API. The consumer might have filtered out the information at that time because it wasn’t in the payroll period that was currently being processed. The API must ensure that the consumer gets the information again when it becomes relevant.

Here’s an example. The initial situation is as follows:

- Job slice 1 begins on January 1, 2014
- Job slice 2 begins on June 15, 2014

In the first case, job slice 2 is deleted. The current payroll period starts on June 1, 2014.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Condition</th>
<th>Does the Condition Apply?</th>
<th>What’s Returned</th>
</tr>
</thead>
<tbody>
<tr>
<td>start_date</td>
<td>start_date &gt;= to_date('2014-06-01', 'YYYY-MM-DD')</td>
<td>The condition is true because it was once true for job slice 2. A consumer might already have received job slice 2 and processed it. Therefore, the consumer working with the start date condition must be informed that the employee no longer has job slice 2.</td>
<td>Job slice 1, including its new end date</td>
</tr>
</tbody>
</table>

The consumer could also get the deletion information using the last_modified_on condition. However, if the deletion was executed in one payroll period but affected a time slice in another payroll period, a consumer might have filtered out this deletion information as it was not relevant at the time of communication. With the start_date condition, the consumer has the chance to get the deletion information at the time when it is relevant.

In the second case, job slice 2 is shifted to either the past or the future.
Shift to the past:
- Job slice 1 begins on January 1, 2014
- Job slice 2 now begins on March 1, 2014

Shift to the future:
- Job slice 1 begins on January 1, 2014
- Job slice 2 now begins on July 1, 2014

The current payroll period starts on June 1, 2014.

Table 7: Job Slice Shifted to Past or Future

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Condition</th>
<th>Does the Condition Apply?</th>
<th>What's Returned</th>
</tr>
</thead>
<tbody>
<tr>
<td>start_date</td>
<td>start_date &gt;= to_date('2014-06-01','YYYY-MM-DD')</td>
<td>The condition is true because it was once true for job slice 2. A consumer might already have received job slice 2 with start date June 15, 2014 and processed it. Therefore, the consumer working with the start date condition must be informed that now job slice 2 starts at another date.</td>
<td>All job slices</td>
</tr>
</tbody>
</table>

This is especially important for shifts to the future. If job slice 2 is shifted to a later payroll period, it must be communicated to the consumer that job slice 2 is no longer effective in the current payroll period. The consumer could also get the shift information using the last_modified_on condition. However, if the shift was executed in one payroll period but affected a time slice in another payroll period (for example a future payroll period), a consumer might have filtered out this shift information as it was not relevant at the time of communication. If the start_date condition did not communicate the shift as well, the consumer might miss the information. Therefore, the start date condition must lead to the selection of the employee for the date where the job slice originally started.

Multiple Identical Responses

Using the start_date condition can lead to the situation that the same API response is returned several times.

Here’s an example. In the first case, the situation is as follows:
- Job slice 1 begins on January 1, 2014
- Job slice 2 begins on June 3, 2014

Job slice 2 was inserted on May 31, 2014. The CompoundEmployee API is called for the payroll period June. Replication takes place on a daily basis.

Table 8: Change Affecting the Same Payroll Period, Multiple Replications for Each Payroll Period

<table>
<thead>
<tr>
<th>API Request</th>
<th>Condition</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>start_date &gt;= to_date('2014-06-01','YYYY-MM-DD')</td>
<td>Employee is returned in the API response. All segments and time slices are returned.</td>
</tr>
</tbody>
</table>
### Table 9: Change Affecting a Future Payroll Period, No Matter If Replication Takes Place Once or Several Times Per Payroll Period

<table>
<thead>
<tr>
<th>API Request</th>
<th>Condition</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>start_date =&gt; to_date('2014-06-01','YYYY-MM-DD')</td>
<td>Employee is returned in the API response. All segments and time slices are returned.</td>
</tr>
<tr>
<td>2</td>
<td>start_date =&gt; to_date('2014-06-02','YYYY-MM-DD')</td>
<td>Employee is returned in the API response. All segments and time slices are returned.</td>
</tr>
<tr>
<td>3</td>
<td>start_date =&gt; to_date('2014-06-03','YYYY-MM-DD')</td>
<td>Employee is returned in the API response. All segments and time slices are returned.</td>
</tr>
<tr>
<td>4</td>
<td>start_date =&gt; to_date('2014-06-04','YYYY-MM-DD')</td>
<td>Employee isn’t returned anymore.</td>
</tr>
</tbody>
</table>

The consumer has adjusted the start date to prevent that the employee is returned with every replication of the current payroll period. Nevertheless, the employee is returned in all replications that take place before the start date of job slice 2.

In the second case, the situation is as follows:

- Job slice 1 begins on January 1, 2014
- Job slice 2 begins on June 15, 2014

Job slice 2 was inserted on May 31, 2014. The CompoundEmployee API is called for the payroll period June. Replication takes place on a daily basis.

Since the change affects a time slice far in the future, the employee is replicated in every payroll run that takes place until the time slice actually gets effective. If the API is called more than once per payroll period, the replication of the same employee data takes place even more often.

In the third case, a change on another segment has already led to replication. Changes can take place on different segments. For example, a new job slice could be inserted with effective start date June 1, 2014 and a spot bonus could be inserted with pay date June 15, 2014. In case of daily replication and start date condition, the employee will be selected on every date between June 1 and June 15. In the first replication, the employee is selected due to the job slice and in the other replications the employee is selected due to the spot bonus. However, since the CompoundEmployee API always returns all data when it runs in full transmission mode, the spot bonus has already been returned with the first replication. Therefore, the API response will look the same in all replications.

This constellation can also occur if replication only takes place once per payroll period. For example, a new job slice is inserted with effective start date June 1, 2014 and a spot bonus is inserted with pay date July 1, 2014. There might be a replication for payroll period June with start date = June 1, 2014 and a replication for payroll
period July with start date = July 1, 2014. The `CompoundEmployee` API response looks the same in both replications. However, the consumer might have ignored the spot bonus information they received on June 1, 2014.

### 2.8.3.25.4 What to Consider When Filtering the Response for the Start Date

If you want to filter the employee information you get from the `CompoundEmployee` API when applying the `START_DATE` condition, you must be aware of the difficulties that can occur when filtering data.

#### Prolonged Time Slices Due to Deletion

The initial situation is as follows:

- Job slice 1 begins on January 1, 2014
- Job slice 2 begins on June 15, 2014

Now, job slice 2 is deleted. Only job slice 1 is left, which begins on January 1, 2014. The current payroll period starts on June 1, 2014.

<table>
<thead>
<tr>
<th>START_DATE Condition</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>start_date &gt;= to_date('2014-06-01','YYYY-MM-DD')</td>
<td>The employee is selected and returned with all their current data, including job slice 1 with its new end date.</td>
</tr>
</tbody>
</table>

**Issues when filtering:** If the consumer filters out the data, there’s a gap from March 15, 2014 through March 30, 2014. Then it seems as if the employee has no job in this time frame. Therefore, it’s important that the consumer doesn’t filter out job slice 1 in the second API request.

#### Prolonged Time Slices Due to Shifts

On January 31, there are two job slices in the system.

- Job slice 1 begins on February 15, 2014
- Job slice 2 begins on March 15, 2014

On February 28, job slice 2 is shifted to the future.

- Job slice 1 begins on February 15, 2014
- Job slice 2 begins on March 30, 2014

The current payroll period is March.
Table 11: Prolonged Time Slice Due to Shift

<table>
<thead>
<tr>
<th>API Request</th>
<th>START_DATE Condition</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>start_date &gt;= to_date('2014-02-01','YYYY-MM-DD')</td>
<td>The employee is returned in the API response. All segments and time slices are returned. Consumer might filter out job slice 2 and memorize March 14, 2014 as end date for job slice 1.</td>
</tr>
<tr>
<td>2 (after the shift of job slice 2)</td>
<td>start_date &gt;= to_date('2014-03-01','YYYY-MM-DD')</td>
<td>The employee is returned in the API response. All segments and time slices are returned. Job slice 2 is sent with start date March 30, 2014. Consumer might filter out job slice 1 because the start date is outside the current payroll period.</td>
</tr>
</tbody>
</table>

Issues when filtering:

- The `CompoundEmployee` API originally sent job slices 1 and 2 to the consumer. However, the consumer might have filtered out job slice 2 and not processed it because at that time the slice wasn’t in the payroll period currently being processed. This means that the consumer might have stored 2004.06.14 as the end date for job slice 1.
- When job slice 2 is deleted, the `CompoundEmployee` API sends the employee again – this time with job slice 1 lasting from January 1, 2014 to December 31, 9999. By doing so, the API informs the consumer that the job doesn’t end on June 14, 2014. However, since the start date hasn’t changed and is before the requested date, the consumer might not see at first glance that this slice is relevant. It’s important that the consumer takes care that they don’t filter out job slice 1 in the given data constellation.

Shortened Time Slices Due to Shifts

On January 31, there are two job slices in the system.

- Job slice 1 begins on February 15, 2014
- Job slice 2 begins on March 15, 2014

On February 28, job slice 2 is shifted to the past.

- Job slice 1 begins on February 15, 2014
- Job slice 2 begins on March 3, 2014

The current payroll period is March.

Table 12: Shortened Time Slices Due to Shifts

<table>
<thead>
<tr>
<th>API Request</th>
<th>START_DATE Condition</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>start_date &gt;= to_date('2014-02-01','YYYY-MM-DD')</td>
<td>The employee is returned in the API response. All segments and time slices are returned. Consumer might filter out job slice 2 and memorize March 14, 2014 as end date for job slice 1.</td>
</tr>
</tbody>
</table>
**Issues when filtering:** If the consumer filters out the data, they must be aware that the shift of job slice 2 has also influenced the end date of job slice 1.

### 2.8.3.26 Extensibility of the CompoundEmployee API

The **CompoundEmployee** API supports two forms of extensibility: Employee Central custom fields and custom MDF objects.

- **Extending the API with Custom Fields** [page 70]
  You can extend the **CompoundEmployee** API with custom fields of HRIS-based objects and MDF-based objects and with custom compositions.

- **Extending the API with MDF Objects** [page 71]
  To extend the **Compound Employee** API with further standard or custom MDF objects, register them so that the API extracts them along with the HRIS objects.

- **Using the Compound Employee API Object Types Object with Legislatively Sensitive Data Configuration** [page 73]
  Look at how to use the **Compound Employee API Object Types** object in combination with a **Legislatively Sensitive Data Configuration** object.

#### 2.8.3.26.1 Extending the API with Custom Fields

You can extend the **CompoundEmployee** API with custom fields of HRIS-based objects and MDF-based objects and with custom compositions.

**Custom Fields and HRIS-Based Objects**

If an HRIS object supports custom fields, the **CompoundEmployee** API considers these custom fields as well. They’re a part of the static fields list **Compound Employee API - Supported Fields**.

Custom fields are contained in the response if the field is not blank or not nil (as for all other fields in the static fields list) and independent from data model or country/region-specific data model.

**Example**

```xml
<customLong1>3</customLong1>
```
Custom Fields and MDF-Based Objects

Custom fields maintained in MDF appear in the response as they’re modeled in the MDF object definition. The following objects are supported:

- paymentinformationV3
- alternative cost distribution
- recurring deduction
- one time deduction
- IT declaration

The following data types for custom fields are ignored by the CompoundEmployee API and excluded from the response:

- Attachments
- Data Sources
- Character large objects (CLOB)

Custom Compositions

Custom compositions to custom objects that are defined in the following MDF objects are supported by the API:

- paymentinformationV3 and child segments
- recurring deductions
- one-time deductions
- alternative cost distribution
- IT declaration

All fields used that are defined in the object definition are returned by the API.

2.8.3.26.2 Extending the API with MDF Objects

To extend the Compound Employee API with further standard or custom MDF objects, register them so that the API extracts them along with the HRIS objects.

Context

Restriction

You aren’t allowed to change the object definition of Employee Central Compound Employee API Object Types.

You can register no more than 20 MDF objects. This includes standard as well as custom objects.
You can register only MDF objects for employee master data. You can't register objects for transactional data, such as employee time data. Objects for transactional data lead to memory issues.

If your custom MDF object supports multiple changes for each day, you can only use it in the full transmission mode of the Compound Employee API.

**Procedure**

1. Go to the Admin Center and choose the Manage Data admin tool.
2. In the Create New field, choose Employee Central Compound Employee API Object Types.
3. Under Object Type, choose the MDF object you want to register.
   
   You can choose only custom MDF objects or standard MDF objects from the allowlist. Currently the following standard objects are in the allowlist:
   
   - Work Order
   - Higher Duty Temp Assignment

   MDF custom object types must fulfill the following requirements to be selectable in Employee Central Compound Employee API Object Types:
   
   - The object is employment-specific. That is, the externalCode field is of data type User.
   - API visibility is set to Read Only or Editable.
   - The object isn’t a composite object.
   - MDF Version History is set to Complete History.

4. Save your changes.

**Next Steps**

After you've set up the object Employee Central Compound Employee API Object Types, the DescribeEx is adjusted accordingly. Now, to extract data from the registered object type, you just have to include the registered MDF object as a select item in your query request.

Don’t change MDF Version History for an object definition after registration, or else the correctness of snapshot, delta, and period delta modes is compromised.

Don’t change the assigned user for an MDF object after registration. If you need to change the user, delete the object instance and create a new object with the new user assignment. Else, delta transmission doesn’t work as expected.

**Related Information**

[Describe and DescribeEx](#) [page 74]
[Query Request Structure](#) [page 20]
### 2.8.3.26.3 Using the Compound Employee API Object Types Object with Legislatively Sensitive Data Configuration

Look at how to use the **Compound Employee API Object Types** object in combination with a **Legislatively Sensitive Data Configuration** object.

The following conditions must be fulfilled when using the **Compound Employee API Object Types** object in combination with **Legislatively Sensitive Data Configuration**. These prerequisites apply whether with or without a **DRTM MDF Custom Purge Objects** object of the same object type that is already registered in the **Compound Employee API Object Types** list.

- In the **Legislatively Sensitive Data Configuration** object definition, `dataSubjectField` must be `externalCode`. `externalCode` must be of type `User`.
- In the **DRTM MDF Custom Purge Objects** object, `Purge Group` must be of type `EMPLOYMENT_INFORMATION`.

**Note**

The `Purge Group` is based on the field `Functional Area` of the purge object’s associated **Legislatively Sensitive Data Configuration**.

- If the custom MDF object is effective-dated, `Date Field of Retention Time` of **DRTM MDF Custom Purge Objects** must be `Effective End Date`. This must be the technical name `mdfSystemEffectiveEndDate`. Please make sure that this date isn’t changed.

**Legislatively Sensitive Data Configuration** and **DRTM MDF Custom Purge Objects** are validated against the custom MDF objects of the same object type that are registered in **Compound Employee API Object Types**. The properties of all objects are checked against each other, no matter which object is changed or created. The conditions are checked if the object type is registered in **Compound Employee API Object Types** and **Legislatively Sensitive Data Configuration** or **DRTM MDF Custom Purge Objects** exist or are created at the same time.

**Caution**

We can’t guarantee that the data returned by the **CompoundEmployee** API is correct if **Legislatively Sensitive Data Configuration** or **DRTM MDF Custom Purge Objects** for the queried custom MDF object are changed after data has been purged. The **CompoundEmployee** API relies on **Legislatively Sensitive Data Configuration** and the configuration of **DRTM MDF Custom Purge Objects** to calculate the response. If the configuration is changed – for example, if a different date field is selected for `Date Field of Retention Time` in **DRTM MDF Custom Purge Objects** – incorrect data can be returned by the API.

### 2.8.4 API Metadata Access

The **CompoundEmployee** API also provides operations that return its metadata.

- **List** [page 74]
  The list operation includes the entity type **CompoundEmployee**.
- **Describe and DescribeEx** [page 74]
The CompoundEmployee API operations describe and describeEx return metadata such as <field> elements for segment names or custom fields.

### 2.8.4.1 List

The list operation includes the entity type CompoundEmployee.

### 2.8.4.2 Describe and DescribeEx

The CompoundEmployee API operations describe and describeEx return metadata such as <field> elements for segment names or custom fields.

The following <field> elements are returned by describe and describeEx:

- Elements for all segment names in the SELECT clause.
- Elements for all selection fields that can be used in the WHERE clause. The <supportedOperator> tags reflect the operators that can be used with the field.
- Elements for all standard fields contained in all segments occurring in the result structure.
- Elements for all custom fields that are enabled in the data model.

The hierarchy of the result structure is reflected by XPath-like path expressions in the <name> elements beneath the <field> elements in the result. For example, the last_name field in the personal_information segment under segment person looks like this:

```xml
<field>
  <name>/person/personal_information/last_name</name>
  ...
</field>
```

For result fields, describe and describeEx return the characteristics name, data type, label, picklist, maximum length, support of the operator in, support of the operator like, and filterable.

For segment fields (the ones specified in the SELECT clause), describeEx returns the characteristics name and selectable.

### Related Information

Example: Describe and DescribeEx [page 165]
2.8.5 How the CompoundEmployee API Reacts to Data Purge

The CompoundEmployee API provides some optional data purge checks, which can be enabled by the consumers.

Master data purge and partial purge of personal data have an impact on the result of the CompoundEmployee API since the API returns less data than before.

For example, if an employee’s master data is completely purged, the API no longer returns any data for this employee. If the data is partially purged – for example, if an entity such as address information is purged – the API doesn’t return any information about the purged records. Even a last modified query doesn’t detect purged employees or partially purged entities since no audit records are created for the purged records.

⚠️ Caution

When using the validations provided by the CompoundEmployee API, make sure that you configure your systems, especially retention times, according to the requirements of the integrations you’re running. Otherwise, there’s a high risk that after data was purged in Employee Central, integrations no longer work and employees can only be replicated again when you adjust the integration.

For example, integrations that use an effective end date filter don’t replicate employee data of employees for which the date of the effective end date filter is in the purge period of any of the requested entities. For such employees, the data replication shows an error.

To resolve such issues, the effective end date filter of the affected integration must be adjusted, or the validation must be disabled altogether. Therefore, make sure that you define retention times of purge objects in close alignment with the effective end date filters used in your integrations. When you use the validation, terminated employees aren’t replicated anymore, as soon as any personal data-related entity such as the email or address is purged. Such employees are always replicated with an error, even if they’re rehired.

2.8.5.1 Data Purge Handling

In full transmission mode, the CompoundEmployee API provides additional information to help the downstream systems interpret data that’s no longer available due to purge and to prevent unintended data loss at consumer side.

Whether the CompoundEmployee API provides additional information about data purge is an optional setting. The consumers must enable it. They have the following options:

- Consumers can request a purge status overview with detailed purge information, using the DRTMPurgeStatusOverview segment.
- Consumers can validate against the effective end date filter using the purgeOptions parameter with validateEffectiveEndDateFilter.

⚠️ Caution

Use either one approach or the other. We recommend that you use DRTMPurgeStatusOverview.
2.8.5.2 Purge Status Overview

Use the DRTMPurgeStatusOverview segment of the CompoundEmployee API if the API is to return detailed information about data purge.

Consumers can expose the purge information stored in the DRTMPurgeStatus MDF object in the response of the CompoundEmployee API. The API returns the purge information from the DRTMPurgeStatus object in the DRTMPurgeStatusOverview segment.

2.8.5.2.1 Enabling Purge Status Overview

Enable the CompoundEmployee API to request information about the purge status, so that the consumer can react on data purge.

Procedure

Add the DRTMPurgeStatusOverview segment to the SELECT clause of the query request.

Related Information

SELECT Clause [page 21]

2.8.5.2.2 Structure of the Purge Status Overview Segment

What the DRTMPurgeStatusOverview segment of the CompoundEmployee API looks like.

The CompoundEmployee API response adds the DRTMPurgeStatusOverview segment at the end of the person segment. The DRTMPurgeStatusOverview segment includes all existing DRTMPurgeStatus subsegments in descending order of the fields node_name and person_id or user_id.

The key fields of the DRTMPurgeStatus subsegment are:

- node_name
- Either user_id or person_id, depending on whether the data was purged for an employment-related object (user_id) or a person-related object (person_id)

Altogether, the DRTMPurgeStatus subsegment has the following fields:

- nodeName
  Contains the name of the CompoundEmployee API segment to which the purge status object belongs.
- highestBusinessPurgeDate
Is filled when an effective-dated object is partially purged for an active employee. It contains the date of the purge execution, minus the retention time. The exact time information is cut off. highestBusinessPurgeDate marks the start of the retention period. All time slices that end before this date is purged.

**Example**

Let's say, data is purged on October 17, 2018 at 08:00:00 00 local time (for example, CET). The retention period is one month. Then highestBusinessPurgeDate is September 17, 2018 in the CompoundEmployee API response. This means, all time slices that end on September 16, 2018 or earlier are purged.

- **highestAuditPurgeDateTime**
  Is filled when audit data is purged. It contains the date and time until which the audit data for the affected segment was purged, in local time. CompoundEmployee API converts the date and time to a UTC time stamp. This matches the exact date and time in UTC when audit data is available again.

**Example**

Let's say, audit data is purged on October 17, 2018 at 08:00:00 00 local time (for example, CET). The retention period for audit data is one month. Then all audit time slices with a last_modified_on date of September 17, 2018 00:00:00 local time or earlier are purged. CompoundEmployee API transforms the date and time into the UTC time stamp 2018-09-16T23:00:00Z.

- **completePurgeDateTime**
  Is filled when data is partially purged for a terminated employment and the termination date is outside the configured retention period of the affected segment for inactive employees. It contains the date and exact local time (for example, CET) when the data was purged. CompoundEmployee API converts the date and time to a UTC time stamp. If a complete master data purge was carried out for an employment, CompoundEmployee API only returns data for the respective employee if another employment exists that hasn’t been fully purged. The purge removes the terminated employment from the database, including all subsegments. But CompoundEmployee API exposes only the complete purge of the employment to the consumer, by rendering a DRTMPurgeStatusOverview segment for the employment information that includes the completePurgeDateTime field.

Consumers can compare the timestamps of completePurgeDateTime and rehiredAtDateTime to identify if the segment needs to be treated currently as purged or active.

- **rehiredAtDateTime**
  Is filled when an employee is rehired. Rehiring into an existing employment adds the rehiredAtDateTime for all previously existing employment-related DRTMPurgeStatus objects. Rehiring into new employment adds only the rehiredAtDateTime to all previously existing person-related DRTMPurgeStatus objects. It contains the date and exact local time (for example, CET) of the specified rehiring, and not the effective start date of the rehiring. Note that rehiredAtDateTime is only written if the rehiring was performed once Employee Central has been updated to release b2011. CompoundEmployee API converts the date and time to a UTC time stamp.

Consumers can compare the time stamps of completePurgeDateTime and rehiredAtDateTime to detect which segment needs to be processed as purged or active.
2.8.5.2.3 Example: Query Response When Requesting Purge Status Overview

What the response returned by CompoundEmployee API looks like if purge status overview is requested.

Sample Code

```xml
<result>
  <sfobject>
    <id>1501</id>
    <type>CompoundEmployee</type>
    <person>
      <person_id>1501</person_id>
      <person_id_external>sgdpr</person_id_external> ...
    </person>
    <DRTMPurgeStatusOverview>
      <DRTMPurgeStatus>
        <node_name>address_information</node_name>
        <person_id>1501</person_id>
        <highestBusinessPurgeDate>2014-12-31</highestBusinessPurgeDate>
        <highestAuditPurgeDateTime>2016-12-30T23:00:00.000Z</highestAuditPurgeDateTime>
        <completePurgeDateTime>2012-06-15T11:52:32.000Z</completePurgeDateTime>
        <rehiredAtDateTime>2013-02-08T08:41:21.000Z</rehiredAtDateTime>
      </DRTMPurgeStatus>
      <DRTMPurgeStatus>
        <node_name>compensation_information</node_name>
        <user_id>sgdpr</user_id>
        <highestBusinessPurgeDate>2014-12-31</highestBusinessPurgeDate>
        <completePurgeDateTime>2012-06-15T11:52:32.000Z</completePurgeDateTime>
        <rehiredAtDateTime>2013-02-08T08:41:21.000Z</rehiredAtDateTime>
      </DRTMPurgeStatus>
      <DRTMPurgeStatus>
        <node_name>dependent_address_information</node_name>
        <person_id>1502</person_id>
        <highestBusinessPurgeDate>2013-12-31</highestBusinessPurgeDate>
        <completePurgeDateTime>2017-11-30T23:00:00.000Z</completePurgeDateTime>
      </DRTMPurgeStatus>
      <DRTMPurgeStatus>
        <node_name>dependent_personal_information</node_name>
        <person_id>1502</person_id>
        <highestBusinessPurgeDate>2013-12-31</highestBusinessPurgeDate>
        <completePurgeDateTime>2017-11-30T23:00:00.000Z</completePurgeDateTime>
      </DRTMPurgeStatus>
      <DRTMPurgeStatus>
        <node_name>email_information</node_name>
        <person_id>1501</person_id>
        <highestAuditPurgeDate>2016-12-30T23:00:00.000Z</highestAuditPurgeDate>
        <completePurgeDateTime>2012-06-15T11:52:32.000Z</completePurgeDateTime>
        <rehiredAtDateTime>2013-02-08T08:41:21.000Z</rehiredAtDateTime>
      </DRTMPurgeStatus>
      <DRTMPurgeStatus>
        <node_name>person_relation</node_name>
        <person_id>1501</person_id>
        <highestBusinessPurgeDate>2013-12-31</highestBusinessPurgeDate>
        <completePurgeDateTime>2017-11-30T23:00:00.000Z</completePurgeDateTime>
      </DRTMPurgeStatus>
    </DRTMPurgeStatusOverview>
  </sfobject>
</result>

Employee Central Compound Employee API

CompoundEmployee API: General Information About Full, Snapshot, and Delta Transmission Modes
2.8.5.2.4 Entities Supporting Purge Status Overview

Purge status overview information is supported for some segments of the CompoundEmployee API, but not for all of them.

Entities Supporting Purge Status Overview

<table>
<thead>
<tr>
<th>Segment or Subsegment</th>
<th>More Info</th>
</tr>
</thead>
<tbody>
<tr>
<td>person</td>
<td></td>
</tr>
<tr>
<td>personal_information</td>
<td></td>
</tr>
<tr>
<td>address_information</td>
<td></td>
</tr>
<tr>
<td>email_information</td>
<td></td>
</tr>
<tr>
<td>phone_information</td>
<td></td>
</tr>
<tr>
<td>Segment or Subsegment</td>
<td>More Info</td>
</tr>
<tr>
<td>-----------------------</td>
<td>-----------</td>
</tr>
<tr>
<td>person_relation</td>
<td>If data is purged for an employee’s dependents, the API returns the purge status for the dependent_information and person_relation segments. This implies that the data of the dependent as well as the relation between the employee and the dependent have been purged. The node_name value differs for the dependent_information and person_relation segments, but the other fields are identical.</td>
</tr>
<tr>
<td>employment_information</td>
<td>Can only be purged if the employment is terminated and a master data purge is executed.</td>
</tr>
<tr>
<td>compensation_information</td>
<td></td>
</tr>
<tr>
<td>paycompensation_recurring</td>
<td>This subsegment can only be purged with the compensation_information segment. That's why it's communicated only with this segment.</td>
</tr>
<tr>
<td>paycompensation_non_recurring</td>
<td></td>
</tr>
<tr>
<td>payment_information</td>
<td></td>
</tr>
<tr>
<td>job_relation</td>
<td></td>
</tr>
<tr>
<td>deduction_recurring</td>
<td></td>
</tr>
<tr>
<td>deduction_non_recurring</td>
<td></td>
</tr>
<tr>
<td>ItDeclaration</td>
<td></td>
</tr>
<tr>
<td>associated_employee_employment_information</td>
<td>For associated employees, the API returns information about a master data purge only. That is, the associated_employee_employment_information child segment of the associated_employee_information segment is returned.</td>
</tr>
<tr>
<td>national_id_card</td>
<td></td>
</tr>
<tr>
<td>dependent_information</td>
<td>If data is purged for an employee’s dependents, the API returns the purge status for the dependent_information and person_relation segments. This implies that the data of the dependent as well as the relation between the employee and the dependent have been purged. The node_name value differs for the dependent_information and person_relation segments, but the other fields are identical.</td>
</tr>
<tr>
<td>dependent_personal_information</td>
<td>For dependents who are also employees, the API returns information about a purge of their personal information.</td>
</tr>
<tr>
<td>dependent_address_information</td>
<td>For dependents who are also employees, the API returns information about a purge of their address information.</td>
</tr>
<tr>
<td>dependent_national_id_card_information</td>
<td>For dependents who are also employees, the API returns information about a purge of their national ID information.</td>
</tr>
<tr>
<td>emergency_contact_primary</td>
<td></td>
</tr>
<tr>
<td>Segment or Subsegment</td>
<td>More Info</td>
</tr>
<tr>
<td>--------------------------------------------------</td>
<td>---------------------------------------------------------------------------</td>
</tr>
<tr>
<td>BenefitsIntegrationOneTimeInfo</td>
<td></td>
</tr>
<tr>
<td>BenefitsIntegrationRecurringInfo</td>
<td></td>
</tr>
<tr>
<td>PriorService</td>
<td></td>
</tr>
<tr>
<td>Any custom MDF object, which is added to</td>
<td></td>
</tr>
<tr>
<td>CompoundEmployee API and can be purged with an</td>
<td></td>
</tr>
<tr>
<td>Employee Central Employment Information purge</td>
<td></td>
</tr>
<tr>
<td>If data is purged for custom entities, the node_name field shows the name of the custom object or custom object segment.</td>
<td></td>
</tr>
</tbody>
</table>

**Segments Not Supporting Purge Status Overview**

For the following segments, no purge information is available because the underlying data can't be purged or because they're technical segments.

<table>
<thead>
<tr>
<th>Segment or subsegment</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>global_accounting_information</td>
<td></td>
</tr>
<tr>
<td>job_information</td>
<td></td>
</tr>
<tr>
<td>alternative_cost_distribution</td>
<td></td>
</tr>
<tr>
<td>accompanying_dependent</td>
<td></td>
</tr>
<tr>
<td>direct_deposit</td>
<td></td>
</tr>
<tr>
<td>personal_documents_information</td>
<td></td>
</tr>
<tr>
<td>EmployeeDataReplicationElement</td>
<td></td>
</tr>
<tr>
<td>DRTMPurgeStatusOverview</td>
<td></td>
</tr>
<tr>
<td>EmpCostAssignment</td>
<td></td>
</tr>
<tr>
<td>HDTempAssignment</td>
<td></td>
</tr>
<tr>
<td>Any custom MDF object, which is added to</td>
<td></td>
</tr>
<tr>
<td>CompoundEmployee API, but can't be purged</td>
<td></td>
</tr>
</tbody>
</table>
2.8.5.3 Effective End Date Filter

Use the `purgeOptions` query parameter of the Compound Employee API if the API is to check whether the effective end date filter (which is also known as Full Transmission Start Date (FTSD) in the standard integrations) is in a period for which data was purged.

2.8.5.3.1 Enabling Validation Against Effective End Date Filter

Enable validation against the effective end date filter in the query request of the CompoundEmployee API, so that the consumer can react on data purge.

Context

Validating against the effective end date filter is useful because the CompoundEmployee API just returns the data that is available in the system for the requested entities. This means that after a partial purge, only the remaining data is returned. The consumer might be provided with less data than before. It’s the responsibility of the consumer to handle this situation correctly.

Procedure

1. Ensure that the `effective_end_date` filter parameter is provided in the query string. Otherwise, the CompoundEmployee API ignores the value of the `purgeOptions` parameter and doesn’t carry out the validation.

2. Enable the validation using the `purgeOptions` query parameter as shown in this example:

```
<urn:query>
  <urn:queryString>
    SELECT person, personal_information, address_information, ...
    FROM CompoundEmployee
    WHERE last_modified_on > to_DateTime('2017-08-01T00:00:00Z')
    AND effective_end_date >= to_date('2016-01-01')
  </urn:queryString>
  <urn:param>
    <urn:name>purgeOptions</urn:name>
    <urn:value>validateEffectiveEndDateTimeFilter</urn:value>
  </urn:param>
</urn:query>
```
Example

The following example shows employees with different retention periods and different time slices. The highest purge date is the day before the begin of the retention period.

Without the validation, the CompoundEmployee API returns all active time slices that are valid on the effective end date filter and beyond. The consumer doesn’t get any information about the purged time slices.

Using the validation, the CompoundEmployee API returns the same result as without the validation for employees 1 and 4. For employees 2 and 3, however, the API returns an error, since the effective end date filter hits a purge period for at least one of the requested entities. For employee 2, for example, the effective end date filter hits the purge period of the address information. For employee 3, it’s the purge period of the spot bonus.

2.8.5.3.2 How the Effective End Date Filter Works

The CompoundEmployee API goes through these steps if validation against the effective end date filter is requested in the query.

Validation against the effective end date filter in the CompoundEmployee API is available for all entities that support partial purge. The steps of the validation are:

1. For each employee and entity, the API determines the highest purge date from the Purge Status MDF object. The highest purge date indicates the date from which on complete data is available for the entity. If
2. If a highest purge date exists, the API checks whether the date is before the effective end date filter.
3. If the highest purge date is before the effective end date filter, the API returns the data of the entity as it is, without an error.
4. If the highest purge date is on or after the effective end date filter, the API proceeds as follows:
   ○ For non-effective-dated entities, the API returns an error for the employee.
   ○ For effective-dated entities, the API checks whether a record exists that is valid at the effective end date filter. If no such record exists, the API returns an error for the employee. Otherwise, it returns the data of the entity without error.

2.8.5.3.3 Example: Query Response When Using Effective End Date Filter

What the response returned by CompoundEmployee API looks like if validation against the effective end date filter is enabled.

The following example shows the response of the CompoundEmployee API for a query with parameter **purgeOptions = validateEffectiveEndDateFilter and effective_end_date >= to_date('2016-12-01')** and two selected employees.

For the first employee, the address information was purged on January 1, 2017 (with the highest purge date December 31, 2016) and no valid record exists on the data of the effective end date filter, December 1, 2016.

That's why the query returns an error for this employee.

The second employee is returned completely since a valid record exists at the date of the effective end date filter.

```
<result>
  <sfobject>
    <id>4711</id>
    <type>CompoundEmployee</type>
    <log>
      <log_item>
        <person_id>4711</person_id>
        <person_id_external>cgrant</person_id_external>
        <code>COMPOUND_EMPLOYEE/EMPLOYEE_ERROR</code>
        <severity>ERROR</severity>
        <message_text>Data for user id cgrant can't be returned:
Please see log items for more information.</message_text>
      </log_item>
      <log_item>
        <person_id>4711</person_id>
        <person_id_external>cgrant</person_id_external>
        <code>COMPOUND_EMPLOYEE/EFFECTIVE_END_DATE_FILTER_IN_PURGE_PERIOD</code>
        <severity>ERROR</severity>
        <message_text>The effective end date filter is outside of
the retention period of address information that starts on 2017-01-01. Please
use an effective end date filter greater than or equal to 2017-01-01.
      </log_item>
    </log>
  </sfobject>
</result>
```
2.8.5.3.4 Entities Supporting Effective End Date Filter

The CompoundEmployee API applies validation against the effective end date filter to all entities that support partial purge.

Entities Supporting Effective End Date Filter

The following entities support the effective end date filter:

<table>
<thead>
<tr>
<th>Entity</th>
<th>Effective-Dated</th>
<th>Base Date Used by Entity</th>
<th>Entity Affects Only Inactive Employees</th>
<th>Entity Supports Complete Purge</th>
</tr>
</thead>
<tbody>
<tr>
<td>personal_information</td>
<td>Yes</td>
<td>end_date</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>address_information</td>
<td>Yes</td>
<td>end_date</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>email_information</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>phone_information</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>person_relation</td>
<td>Yes</td>
<td>end_date</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>compensation_information</td>
<td>Yes</td>
<td>end_date</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Entity</td>
<td>Effective-Dated</td>
<td>Base Date Used by Entity</td>
<td>Entity Affects Only Inactive Employees</td>
<td>Entity Supports Complete Purge</td>
</tr>
<tr>
<td>--------------------------------</td>
<td>-----------------</td>
<td>--------------------------</td>
<td>----------------------------------------</td>
<td>--------------------------------</td>
</tr>
<tr>
<td>alternative_cost_distribution</td>
<td>Yes</td>
<td>effectiveEndDate</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>paycompensation_non_recurring</td>
<td>No</td>
<td>pay_date</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>job_relation</td>
<td>Yes</td>
<td>end_date</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>deduction_recurring</td>
<td>Yes</td>
<td>effectiveEndDate</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>deduction_non_recurring</td>
<td>No</td>
<td>deductionDate</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>ItDeclaration</td>
<td>Yes</td>
<td>effectiveEndDate</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>PaymentInformationV3</td>
<td>Yes</td>
<td>effectiveEndDate</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>national_id_card</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>emergency_contact_secondary</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

The validation is also applied to global assignments and concurrent employment data that are purged in master data purge:

Table 13: Global Assignment and Concurrent Employment Entities Supporting Effective End Date Filter

<table>
<thead>
<tr>
<th>Entity</th>
<th>Used Base Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>employment_information</td>
<td>end_date</td>
</tr>
<tr>
<td>associated_employee_information</td>
<td>end_date</td>
</tr>
</tbody>
</table>

**CompoundEmployee API Entities and Corresponding Purge Objects**

The following table shows the relation of CompoundEmployee API entity and purge object, which is defined in Data Retention Management:

<table>
<thead>
<tr>
<th>Entity</th>
<th>Entity Belongs to Data Retention Group</th>
<th>Entity Uses Purge Object</th>
<th>Entity Uses Subject ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>personal_information</td>
<td>Person Information</td>
<td>DRTM_PERSONAL_DETAILS</td>
<td>person_id</td>
</tr>
<tr>
<td>address_information</td>
<td>Person Information</td>
<td>DRTM_ADDRESS</td>
<td>person_id</td>
</tr>
<tr>
<td>Entity</td>
<td>Entity Belongs to Data Retention Group</td>
<td>Entity Uses Purge Object</td>
<td>Entity Uses Subject ID</td>
</tr>
<tr>
<td>--------------------------</td>
<td>----------------------------------------</td>
<td>--------------------------</td>
<td>------------------------</td>
</tr>
<tr>
<td>email_information</td>
<td>Person Information</td>
<td>DRTM_EMAIL</td>
<td>person_id</td>
</tr>
<tr>
<td>phone_information</td>
<td>Person Information</td>
<td>DRTM_PHONE</td>
<td>person_id</td>
</tr>
<tr>
<td>person_relation</td>
<td>Person Information</td>
<td>DRTMDEPENDENTS</td>
<td>person_id</td>
</tr>
<tr>
<td>compensation_information</td>
<td>Employment</td>
<td>DRTM_COMPENSATION</td>
<td>user_id</td>
</tr>
<tr>
<td>alternative_cost_distribution</td>
<td>Employment</td>
<td>DRTM_COST_DISTRIBUTION</td>
<td>user_id</td>
</tr>
<tr>
<td>paycompensation_non_recurring</td>
<td>Employment</td>
<td>DRTM_NON_RECURRING_PAY</td>
<td>user_id</td>
</tr>
<tr>
<td>job_relation</td>
<td>Employment</td>
<td>DRTM_JOB_RELATIONSHIPS</td>
<td>user_id</td>
</tr>
<tr>
<td>deduction_recurring</td>
<td>Employment</td>
<td>DRTM_DEDUCTION</td>
<td>user_id</td>
</tr>
<tr>
<td>deduction_non_recurring</td>
<td>Employment</td>
<td>DRTM_DEDUCTION</td>
<td>user_id</td>
</tr>
<tr>
<td>ItDeclaration</td>
<td>Employment</td>
<td>DRTM_INCOME_TAX_DECLARATION</td>
<td>user_id</td>
</tr>
<tr>
<td>PaymentInformationV3</td>
<td>Employment</td>
<td>DRTM_PAYMENT_INFORMATION</td>
<td>user_id</td>
</tr>
<tr>
<td>national_id_card</td>
<td>Person Information</td>
<td>DRTM_NATIONAL_ID_CARD</td>
<td>person_id</td>
</tr>
<tr>
<td>emergency_contact_primary</td>
<td>Person Information</td>
<td>DRTM_EMERGENCY_CONTACT_INFO</td>
<td>person_id</td>
</tr>
</tbody>
</table>

The validation is also applied to custom MDF objects that support partial purge and that are configured according to **Legislatively Sensitive Data Configuration**. In this case, the MDF object name will be used as purge object.

### 2.8.5.3.5 Partial Purge of Inactive Employees

For terminated employees, partial purge will purge the complete data of an entity as soon as the termination date is outside of the retention period of the entity. If the data of an entity is purged completely, the standard effective end date validation doesn’t work.

That’s why a different handling is required here:

- The **CompoundEmployee API** introduces a new error code that indicates complete purge of an entity.
- The API returns an error with this code for all employees for which at least one person-related entity, such as email or address information, was completely purged.
The API ignores all employments of an employee for which at least one employment-related entity, such as compensation information, was completely purged. If all of the employee’s employments are affected by complete purge, the API returns an error for this employee with the new error code. The new error code enables consumers to detect employees with completely purged entities and to react accordingly.

例

The standard integration we provide for replicating employee master data from Employee Central to ERP systems ignores such employees and treats them as successfully replicated.

The complete purge is supported by almost all purgeable entities. It is applied to non-effective dated entities, such as email or phone, as well as to effective-dated entities, such as address information or compensation information. Only personal information is excluded from this handling, since at least the name of the employee should be kept for identification.

The following example shows the purge of address information and email of an inactive employee whose termination date was mid of 2013. Since the retention period of both entities is one year, they were purged completely with the purge run executed in 2015. Personal information is not purged since this entity is excluded from complete purge. The termination date of the employment (dotted line) is now outside of the retention period of email, personal, and address information. Email and address information are purged, whereas personal information remains.

图 4: 示例: 仅删除地址信息和电子邮件

在这个示例中，CompoundEmployee API 将返回以下响应消息：

```xml
<result>
  <sfobject>
    <id>4711</id>
    <type>CompoundEmployee</type>
  </sfobject>
</result>
```
2.8.6 Error Handling in the CompoundEmployee API

How the CompoundEmployee API handles error situations.

The API handles errors depending on the situation in which they occurred:

- Errors aborting the overall API processing
  This can be caused by invalid provisioning settings, general system issues, or an invalid SFAPI request. As a result, a fault response is returned and the corresponding error messages are written into the API Audit Log.
- Errors leading to incomplete data in the response
This can be caused by corrupt business data or misconfigured foundation data. In this case, messages describing the error are appended as child elements to the relevant `<sfobject>` segment of the query response.

**Errors Aborting the Overall API Processing [page 90]**

When errors are caused by invalid provisioning settings, general system issues, or an invalid SFAPI request, the `CompoundEmployee` API returns a fault response.

**Errors Leading to Incomplete Data in the Response [page 90]**

Inconsistencies in application data, foundation objects, or picklists don’t terminate the `CompoundEmployee` API call. Instead, if possible, the API returns as much valid data as possible and adds error messages to show you what went wrong.

**Employee Central Application Message Error Codes [page 94]**

Find out about possible `CompoundEmployee` API error codes and what they mean.

### 2.8.6.1 Errors Aborting the Overall API Processing

When errors are caused by invalid provisioning settings, general system issues, or an invalid SFAPI request, the `CompoundEmployee` API returns a fault response.

If processing is aborted, the API writes corresponding error messages into the API Audit Log. For example:

- System setup issues
- Succession model could not be read for...
- Provisioning API settings missing
- Provisioning audit log not activated
- Service temporarily unavailable
- Invalid query string...

### 2.8.6.2 Errors Leading to Incomplete Data in the Response

Inconsistencies in application data, foundation objects, or picklists don’t terminate the `CompoundEmployee` API call. Instead, if possible, the API returns as much valid data as possible and adds error messages to show you what went wrong.

**Log Item in the Query Response [page 91]**

If the response is incomplete, the `CompoundEmployee` API appends messages describing the error as child elements to the corresponding `<sfobject>` segment of the query response.

**Data Inconsistencies in the Query Response [page 93]**

Inconsistencies in application data, foundation objects, or picklists don’t terminate the `CompoundEmployee` API call. Instead, the API returns as much valid data as possible and adds error messages to show you what went wrong.
2.8.6.2.1 Log Item in the Query Response

If the response is incomplete, the CompoundEmployee API appends messages describing the error as child elements to the corresponding sfobject segment of the query response.

The <log> tag in the query response of CompoundEmployee API contains all messages belonging to a particular employee record. The field that caused the system message remains empty and isn’t returned by the query response.

Log Item

The <log> item in the API response helps the API consumer to evaluate error relevance, severity, and further processing, which can be, for example:

- Process the employee records without errors.
- Filter employee records with errors and raise an alert for affected users or administrators.
- Ignore system message because the employee field isn’t relevant.
- Use log item to trigger forward error handling in receiving systems.

The <log> tag contains the following subtags:

- <log_item>: There’s one <log_item> for each field that caused an error. Multiple error messages for each field are not supported.
- <node_name>: The name of the node (or segment) that contains the erroneous field.
- <field_name>: The hris field name of the erroneous field in the data model.
- <xPath>: This XPath can be used by the consumer to locate the segment with the erroneous field. It can be applied to the query response as a whole.
- <message_text>: The user-friendly error message.
- <severity>: Indicates whether the message is an information, a warning, or an error. Warnings are given, for example, if external codes can’t be determined for certain fields. Severity ERROR is set if the data of an employee can’t be returned at all due to inconsistent data.
- <code>: The stable identifier of the error.
- <person_id>: The person ID of the employee. <person_id> is only used when whole segments are inconsistent.
- <person>: The external person ID of the employee. <person> is only used when whole segments are inconsistent.
- <per_person_uuid>: The person UUID of the employee. <per_person_uuid> is only used when whole segments are inconsistent.

The consumer must ensure that the root cause is fixed in the configuration. This can be done, for example, by the middleware, which publishes the errors in an RSS feed to an administrator.

Once the root cause has been fixed, the affected employee records can be synchronized again. Either in the middleware by triggering the Employee Central query for affected person IDs, or from the Employee Central UI by applying a dummy change to the failed employee record. The preferred approach is to synchronize the employee records from the Employee Central UI.
Log Items Appended as Child Tags to the Query Response

The `<log>` tag contains all messages belonging to a particular employee record. The field that caused the system message remains empty and isn’t returned by the API response.

Examples for error situations:

- External codes for picklists or foundation objects are missing.
- Picklist lookup failed due to an invalid option ID.

---

Sample Code

```xml
<queryResponse>
  <result>...
    <sfobject>
      <person>
        <!-- all fields filled except those which raised log items -->
        </person>
        <!-- every sfoobject contains exactly one person, hence all log items refer to the same person -->
        <log>
          <log_item>
            <node_name>employment_information</node_name>
            <field_name>emplStatus</field_name>
            <severity>Warning</severity>
            <message_text>
              "<Picklist value>" can't be returned. Please make sure that the external code for picklist "<picklist name>" with ID "<external code>" is valid.
            </message_text>
          </log_item>
        </log>
    </sfobject>
  </result>
</queryResponse>
```

---

Log Items for Failed Employee Queries

There are situations where data inconsistencies prevent the API from returning any data for a particular person at all. In these situations only a log item containing error information, but no data besides the ID and external ID are returned.

---

Sample Code

```xml
<log>
  <log_item>
    <person_id>220</person_id>
    <person_id_external>KJACOB</person_id_external>
    <severity>Error</severity>
    <message_text>
      Data for user ID KJACOB can't be returned. Please provide a legal entity for job with start date 2018-10-01.
    </message_text>
  </log_item>
</log>
```
The following situations are handled by such a log item:

- **No valid legal entity**
  
  No valid legal entity is found for job or compensation time slices. This can happen if the start date of the job or compensation is before the start date of the legal entity. As some fields in the job and compensation segments are determined by the legal entity, processing these segments without the legal entity isn’t possible.

- **No legal entity is assigned to a job**
  
  No legal entity is assigned to job. As some fields in the job and compensation segments are determined by the legal entity, processing these segments without the legal entity isn’t possible.

- **Initial start of job and compensation differs**
  
  No valid job is found for a compensation time slice. This can happen if the start date of the compensation is before the start date of the job with event type **Hire**.

---

### i Note

The first time slice of job and compensation must have the same start date. Subsequent time slices can differ.

---

### Other Internal Errors

An internal error appears that isn’t handled by a specific error message.

---

### 2.8.6.2.2 Data Inconsistencies in the Query Response

Inconsistencies in application data, foundation objects, or picklists don’t terminate the CompoundEmployee API call. Instead, the API returns as much valid data as possible and adds error messages to show you what went wrong.

---

### Inconsistencies Regarding Time Slices on Job and Foundation Objects

The following more complex data inconsistencies prevent the API from uniquely determining the legal entity belonging to a job time slice:

- **Overlapping time slices for an event reason that is assigned to an employee’s job, when the start date of a job time slice is within the overlapping time frame of the event reason.**

- **Several overlapping time slices of a legal entity that is assigned to an employee’s job, when the effective start date of the job or compensation time slice is in the overlapping time frame of the legal entity slices.**

- **An employee has several overlapping job time slices.**
Inconsistencies Regarding Start Dates and Sequence Numbers

On the job_information and compensation_information segments, it’s possible to maintain various records for the same time frame. These records are differentiated from each other by sequence numbers. The combination of start date and sequence number must be unique for each employee. If employee data contains two records with the same start date and same sequence number, the CompoundEmployee API reacts differently depending on whether these records are part of the response or not.

The following cases result in an error message:

- Inconsistency on the job_information or compensation_information segment on the highest sequence number
- Inconsistency on the job_information segment on a sequence number that isn’t the highest sequence number, but parameter resultOptions with value allJobChangesPerDay was requested
- Inconsistency for compensation_information if duplicate pay components exist
- Inconsistency on the job_information or compensation_information segment on a sequence number that isn’t the highest sequence number, but parameter segmentsForEnhancedEffectiveEndDateFilter with value job_information or compensation_information was requested.

**i Note**

The segmentsForEnhancedEffectiveEndDateFilter parameter can be used if the effective_end_date parameter is included in the query request and the resultOptions parameter has one of the values allJobChangesPerDay or allCompensationChangesPerDay.

The following inconsistencies don’t result in an error message:

- Inconsistency on the job_information or compensation_information segments on the sequence number, which isn’t the highest sequence number.
- Inconsistency on a job_information or compensation_information record that is filtered out due to effective_end_date selection.

### 2.8.6.3 Employee Central Application Message Error Codes

Find out about possible CompoundEmployee API error codes and what they mean.

#### Error Code Suffixes

A log item <log> tag contains the subtag <code>, which is a stable identifier of the error returned by the API. An error code is made up of the following parts: COMPOUND_EMPLOYEE/<Error Code Suffix>. The table lists error code suffixes in alphabetical order and gives a short description of what they mean.
<table>
<thead>
<tr>
<th>Error Code Suffix</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ASSIGNED_MANAGER_IS_INVALID</td>
<td>The manager of the employee can't be found. That's why the fields manager_person_id, manager_employment_id and manager_person_id_external can't be returned.</td>
</tr>
<tr>
<td>COMPENSATION_BEFORE_JOB_ERROR</td>
<td>The start date of a person's compensation is earlier than the start date of that person's job.</td>
</tr>
<tr>
<td>COMPLETE_ENTITY_PURGE</td>
<td>The data of at least one entity was purged completely. No data is returned.</td>
</tr>
<tr>
<td>COST_CENTER_MAPPING_EXTERNAL_CODE_ERROR</td>
<td>Cost center key mapping can't be applied because the external code for the affected cost center isn't valid.</td>
</tr>
<tr>
<td>COST_CENTER_MAPPING_INTERNAL_CODE_ERROR</td>
<td>Cost center key mapping can't be applied because no cost center exists with the internal code at the specified date.</td>
</tr>
<tr>
<td>COST_CENTER_MAPPING_NO_EXTERNAL_KEY_ERROR</td>
<td>Cost center key mapping can't be applied because there’s no external key maintained for the cost center.</td>
</tr>
<tr>
<td>COUNTRY_NOT_FOUND_LEGAL_ENTITY</td>
<td>Country/region can't be determined for the legal entity.</td>
</tr>
<tr>
<td>DATE_EXCEEDS_RESTRICTION</td>
<td>The last_modified_on or snapshot_date exceeds the restriction of 3 months into the past.</td>
</tr>
<tr>
<td>DELTA_COMPLEX_CHANGE_ERROR</td>
<td>The last changes to the employee for a certain entity can’t be extracted. Manual replication is required.</td>
</tr>
<tr>
<td>DELTA_DUPLICATE_KEYS</td>
<td>Delta can't be calculated for a certain entity because there are multiple records with the same business key.</td>
</tr>
<tr>
<td>DELTA_DUPLICATE_SEQUENCE_NUMBER</td>
<td>Delta can't be calculated for a certain entity because there are multiple records with the same sequence number on the same date.</td>
</tr>
<tr>
<td>DELTA_ENTITY_ERROR</td>
<td>General error that delta can’t be calculated for a certain entity because the entity has an error.</td>
</tr>
<tr>
<td>DELTA_INCONSISTENT_END_DATE</td>
<td>Delta can’t be calculated for a certain entity because its end date is earlier than its start date.</td>
</tr>
<tr>
<td>DELTA_INCONSISTENT_TIME_SLICES</td>
<td>General error that delta can’t be calculated for a certain entity because the entity has inconsistencies.</td>
</tr>
<tr>
<td>DELTA_INVALID_PERIOD</td>
<td>An invalid period is given for the calculation of period delta, for example, the end date isn’t available from the request.</td>
</tr>
<tr>
<td>DELTA_MULTIPLE_RECORDS_AT_START_DATE</td>
<td>Delta can’t be calculated for an entity because it has multiple records on the given start date.</td>
</tr>
<tr>
<td>Error Code Suffix</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------------------------------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>DELTA_NO_RELEVANT_CHANGE</td>
<td>The employee has changed since the last replication but the changes aren’t relevant, for example, the value hasn’t changed between last and current replication. No data is turned for the employee.</td>
</tr>
<tr>
<td>DELTA_NO_RELEVANT_CHANGE_IN_PERIOD</td>
<td>The employee has changed since the last replication but the changes aren’t relevant. For example, the value hasn’t changed during the chosen period. No data is returned for the employee.</td>
</tr>
<tr>
<td>DUPLICATE_KEYS_ERROR</td>
<td>There are multiple records with the same business key.</td>
</tr>
<tr>
<td>DUPLICATE_PAY_COMPONENT_ERROR</td>
<td>A compensation contains more than one pay component record.</td>
</tr>
<tr>
<td>DUPLICATE_SEQUENCE_NUMBER_ERROR</td>
<td>A given entity has multiple records with the same sequence number on the same day.</td>
</tr>
<tr>
<td>EFFECTIVE_END_DATE_FILTER_IN_PURGE_PERIOD</td>
<td>The effective end date filter is outside of the retention period of at least one requested entity.</td>
</tr>
<tr>
<td>EMPLOYEE_ERROR</td>
<td>General error that an employee can’t be returned because of a failure. Another log item is placed in the request that describes the root cause.</td>
</tr>
<tr>
<td>EMPLOYEE_PROCESSING_ERROR</td>
<td>An internal error occurred during processing of the employee. The employee was selected but no data could be retrieved.</td>
</tr>
<tr>
<td>ENTITY_ERROR</td>
<td>Data of a given entity can’t be returned.</td>
</tr>
<tr>
<td>EXTENSION_FIELD_EVALUATION_ERROR</td>
<td>The value of the extension field can’t be determined because the evaluation of the target field failed.</td>
</tr>
<tr>
<td>EXTENSION_FIELD_INVALID_DEFINITION</td>
<td>The extensions field isn’t defined correctly.</td>
</tr>
<tr>
<td>EXTENSION_FIELD_NOT_UNIQUE_VALUE</td>
<td>The value of the extension field can’t be determined because the path is ambiguous.</td>
</tr>
<tr>
<td>FOUNDATION_OBJECT_INVALID</td>
<td>The external code of a given foundation object is invalid.</td>
</tr>
<tr>
<td>FREQUENCY_INVALID</td>
<td>There’s no valid frequency foundation object for a given entity at a given date.</td>
</tr>
<tr>
<td>FROM_DATE_IN_PURGE_PERIOD</td>
<td>The fromDate is outside of the retention period of at least one entity.</td>
</tr>
<tr>
<td>GENERIC_OBJECT_INVALID</td>
<td>The external code of a given generic object is invalid.</td>
</tr>
<tr>
<td>Error Code Suffix</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------------------------------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>GENERIC_OBJECT_INVALID_DEFINITION</td>
<td>There’s no object definition for a given generic object. Please refresh the MDF metadata cache.</td>
</tr>
<tr>
<td>GENERIC_OBJECT_READ_ERROR</td>
<td>An error occurred while trying to read a given generic object.</td>
</tr>
<tr>
<td>GENERIC_RUNTIME_ERROR</td>
<td>An unexpected runtime error occurred.</td>
</tr>
<tr>
<td>INCOMPLETE_EMPLOYEE_DATA</td>
<td>A person is missing Job Information, Personal Information, or Compensation Information. This data isn’t returned.</td>
</tr>
<tr>
<td>INTERNAL_ERROR</td>
<td>An internal error occurred.</td>
</tr>
<tr>
<td>LAST_MODIFIED_ON_IN_AUDIT_PURGE_PERIOD</td>
<td>The last modified on date is outside of the audit retention period of at least one entity.</td>
</tr>
<tr>
<td>LEGAL_ENTITY_NOT_FOUND</td>
<td>A given legal entity is invalid.</td>
</tr>
<tr>
<td>LEGAL_ENTITY_NOT_FOUND_FOR_START_DATE</td>
<td>A given legal entity is invalid for a given start date.</td>
</tr>
<tr>
<td>NO_RELEVANT_EMPLOYEE_DATA</td>
<td>The employee was selected, however, no data was returned because the employee doesn’t have relevant data.</td>
</tr>
<tr>
<td>OVERLAPPING_INTERNAL_ERROR</td>
<td>A record of a given entity overlaps in a time period.</td>
</tr>
<tr>
<td>OVERLAPPING_TIME_SLICE_ERROR</td>
<td>A record of a given entity overlaps in a time period.</td>
</tr>
<tr>
<td>PERIOD_GAP_AT_END_ERROR</td>
<td>The end date of the last record of a given entity isn’t infinite. The record doesn’t have the end date December 31, 9999.</td>
</tr>
<tr>
<td>PERIOD_INCONSISTENT_END_DATE_ERROR</td>
<td>The end date of the record of a given entity is before its start date.</td>
</tr>
<tr>
<td>PERIOD_NO_VALID_RECORD_ERROR</td>
<td>There’s no valid record in the given period.</td>
</tr>
<tr>
<td>PERIOD_OVERLAP_AT_START_DATE_ERROR</td>
<td>There are multiple records with the same start date.</td>
</tr>
<tr>
<td>PERIOD_OVERLAP_ERROR</td>
<td>The records overlap in a given period.</td>
</tr>
<tr>
<td>PERSON_ERROR</td>
<td>General error that a person can’t be returned because of a failure. Another log item is placed in the request that describes the root cause.</td>
</tr>
<tr>
<td>PICKLIST_INVALID</td>
<td>The non-unique external code of a given picklist option isn’t valid or wasn’t maintained.</td>
</tr>
<tr>
<td>RECORD_INVALID</td>
<td>General error that a record can’t be returned.</td>
</tr>
<tr>
<td>SEVERE_PURGE_PERIOD_ERROR</td>
<td>The effective end date filter is outside of the retention period for most of the selected employees. The query is aborted.</td>
</tr>
<tr>
<td>Error Code Suffix</td>
<td>Description</td>
</tr>
<tr>
<td>------------------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>SFQL_EMPTY_ENTITY_SELECTION</td>
<td>Segment and field selection: The expressions for a stated path contain only excluded fields. The expressions are ignored.</td>
</tr>
<tr>
<td>SFQL_INVALID_ENTITY</td>
<td>Segment and field selection: The expression is invalid and is ignored.</td>
</tr>
<tr>
<td>SFQL_INVALID_EXPRESSION</td>
<td>Segment and field selection: The stated element of the expression is invalid. The expression is ignored.</td>
</tr>
<tr>
<td>SFQL_INVALID_FIELD</td>
<td>Segment and field selection: The stated field isn’t defined in the segment. The field is ignored.</td>
</tr>
<tr>
<td>SNAPSHOT_DATE_IN_AUDIT_PURGE_PERIOD</td>
<td>The snapshot date is outside of the audit retention period of at least on requested entity.</td>
</tr>
</tbody>
</table>

**Severity Levels**

Generally, a log item or error code isn’t bound to a specific severity level. Depending on the situation, the severity can differ. The severity itself signifies the amount of erroneous data of the affected person.

<table>
<thead>
<tr>
<th>Severity</th>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>ERROR</td>
<td>No data is returned because the error affects the person as a whole. In this case, the CompoundEmployee API will additionally return a message with the code EMPLOYEE_ERROR or PERSON_ERROR for each affected person.</td>
</tr>
<tr>
<td>WARNING</td>
<td>At least one field is missing. Data that isn’t affected isn’t sent to the consumer.</td>
</tr>
<tr>
<td>INFORMATION</td>
<td>Processing was successful. The message contains more information about the response.</td>
</tr>
</tbody>
</table>
3 Compound Employee API: Snapshot Mode

When it runs in the snapshot mode, the CompoundEmployee API returns the data as it was on the snapshot date, considering all changes, corrections, and deletions.

Enabling the Snapshot Mode

To enable the snapshot mode, set the query parameter `queryMode` to `snapshot` and include the subexpression `snapshot_date = ...` in the `WHERE` condition. The calculation is then done on the basis of the audit records. If for example, the email address of an employee has changed and the snapshot date in the query is before the last modified date of the change, the API returns the employee with the old email address. If the snapshot date is after the last modified date, the API returns the new email address.

Segments Not Supported in Snapshot Mode

The following segments aren't supported for the snapshot mode:

- `payment_information`, because this segment doesn't provide audit information in Employee Central
- `accompanying_dependent`, because this segment doesn't have an HRIS element model

Changes Not Considered in Snapshot Mode

The following types of changes aren't considered by the snapshot mode:

- Changes to external codes of picklists, foundation objects, or generic objects. The current value of the external code is returned.
- Changes to foundation objects that are triggered using `Make Correction`. The current value of the determined time slice of the foundation object is returned.
- Changes to attributes of the objects `USERS_SYS_INFO` and `USER_ACCOUNT`. The current value is returned for the following attributes:
  - `logon_user_id`, `logon_user_name`, and `logon_user_is_active` in the `person` segment.
  - `direct_reports` in the `employment_information` segment.

Data Purge Handling in Snapshot Mode [page 100]

Data Retention Management allows purging of transactional data and audit data independently, with different retention periods. For that reason, the CompoundEmployee API must be able to handle situations where transactional data was purged and audit data is still there.
### 3.1 Data Purge Handling in Snapshot Mode

Data Retention Management allows purging of transactional data and audit data independently, with different retention periods. For that reason, the `CompoundEmployee` API must be able to handle situations where transactional data was purged and audit data is still there.

In snapshot mode, the `CompoundEmployee` API determines the retention times of transactional data for each employee and entity. All records of the snapshot image that are outside of the respective retention period of the underlying entity will be ignored. Snapshot is only calculated for the records that are valid in the retention period. For the period in which data was purged and for the following day, no snapshot is calculated.

The API also checks for each entity whether the provided `snapshot_date` is within the audit retention time of the entity. If this is not the case for one or more entities, the `CompoundEmployee` API returns an error for the employee, indicating that the provided date is not allowed.

**Example**

```xml
<log>
  <log_item>
    <person_id>4711</person_id>
    <person_id_external>Steve</person_id_external>
    <code>COMPOUND_EMPLOYEE/EMPLOYEE_ERROR</code>
    <severity>ERROR</severity>
    <message_text>
      Data for user id Steve can't be returned: Please see log items for more information.
    </message_text>
  </log_item>
  <log_item>
    <person_id>4711</person_id>
    <person_id_external>Steve</person_id_external>
    <code>COMPOUND_EMPLOYEE/SNAPSHOT_DATE_IN_AUDIT_PURGE_PERIOD</code>
    <severity>ERROR</severity>
    <message_text>
      The provided snapshot_date is outside of the audit retention period of entity "phone_information" that starts on 2016-12-30T23:00:00.000Z. Please use a snapshot_date later than 2016-12-30T23:00:00.000Z.
    </message_text>
  </log_item>
</log>
```

The audit retention time of phone information is configured to 6 months. The audit purge was executed for this employee on June 30, and the audit records of all phone information changes prior to December 30...
were deleted. The CompoundEmployee API will not provide a snapshot for this employee that has a snapshot_date before December 30.
4 CompoundEmployee API: Delta Transmission Mode

What you should know when using the delta transmission mode of the CompoundEmployee API.

**Notation for Examples [page 103]**
Within this documentation, we use a specific notation for the visualization of time slices of the snapshot data, the current data, and the calculated delta of an effective-dated entity.

**Delta Transmission Query Request [page 103]**
Find out about how the query request of the CompoundEmployee API call is structured.

**Delta Transmission Query Response [page 116]**
Here’s what the query response of the CompoundEmployee API call looks like.

**Action Codes [page 119]**
The delta transmission mode of the CompoundEmployee API supports some action codes.

**Previous Fields [page 120]**
In case a field has been changed since the last modified date, a subelement called previous is added to the field by the CompoundEmployee API.

**Event Information [page 121]**
Job information and compensation information require special treatment because they contain the information about the business events, for example, a newly-hired employee.

**Effective-Dated and Period-Based Delta Transmission [page 128]**
The CompoundEmployee API offers two different kinds of delta transmission: effective-dated and period-based.

**How the CompoundEmployee API Delta Transmission Mode Handles Foundation Objects [page 152]**
Here are some things you should know about how the delta transmission mode of the CompoundEmployee API handles foundation objects.

**How the CompoundEmployee API Delta Transmission Mode Reacts to Data Purge [page 155]**
Data Retention Management allows purging of transactional data and audit data independently, with different retention periods. That’s why the CompoundEmployee API must be able to handle situations where transactional data was purged and audit data is still there.

**Special Handling for the CompoundEmployee API Delta Transmission Mode [page 157]**
Look at some behaviors that apply to the delta transmission mode of the CompoundEmployee API.
4.1 Notation for Examples

Within this documentation, we use a specific notation for the visualization of time slices of the snapshot data, the current data, and the calculated delta of an effective-dated entity.

Let's look at an example:

The snapshot data and the current data each have two time slices.

The first time slice of the snapshot data has value A. The second time slice has value B.

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>B</td>
</tr>
</tbody>
</table>

**Figure 5: Snapshot Data**

The first time slice of the current data has value C instead of A, which means that the value of the time slice was changed from A to C. The second time slice was not changed and still has value B. The changed time slice is shown in red color for better visualization.

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>C</td>
<td>B</td>
</tr>
</tbody>
</table>

**Figure 6: Current Data**

The delta is calculated by the delta processor on the basis of the snapshot data and the current data. The first time slice shows Change C (A→C), which means that the calculated action code for the time slice is CHANGE and the values were changed from A to C, where C is the current value.

Here, the changed time slice is shown in green color.

<table>
<thead>
<tr>
<th>Change C (A→C)</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>B</td>
</tr>
</tbody>
</table>

**Figure 7: Delta to Be Transferred**

4.2 Delta Transmission Query Request

Find out about how the query request of the CompoundEmployee API call is structured.

The request has the following properties.

- **queryString**
  This is the SELECT statement following SFQL grammar.
- **param**
  This is a list of optional parameters.

The query string is made up of a SELECT clause, a FROM clause, and a WHERE clause, as shown in the example.

```xml
<queryString>
    SELECT <select_items>
    FROM   CompoundEmployee
    WHERE  <expression>
</queryString>
```
SELECT items and the expressions are case-insensitive and are considered as long as the spelling is correct.

4.2.1 SELECT Items

The SELECT items are a list of all entities to be returned as part of the hierarchical query response XML by the delta transmission mode of the CompoundEmployee API.

Supported Substructures

The following substructures are supported in the delta transmission mode.

<table>
<thead>
<tr>
<th>Segment</th>
<th>Supported In Effective-Dated Delta</th>
<th>Supported In Period-Based Delta</th>
<th>Supported in Delta Since</th>
</tr>
</thead>
<tbody>
<tr>
<td>person</td>
<td>Yes</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>personal_information</td>
<td>Yes</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>address_information</td>
<td>Yes</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>email_information</td>
<td>Yes</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>phone_information</td>
<td>Yes</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>person_relation</td>
<td>Yes</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>national_id_card</td>
<td>Yes</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>dependent_information</td>
<td>Yes</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>emergency_contact_primary</td>
<td>Yes</td>
<td>Yes</td>
<td>Q2 2016</td>
</tr>
<tr>
<td>employment_information</td>
<td>Yes</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>global_assignment_information</td>
<td>Yes</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>job_information</td>
<td>Yes</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>alternative_cost_distribution</td>
<td>Yes</td>
<td>Yes</td>
<td>October 11, 2015</td>
</tr>
<tr>
<td>compensation_information</td>
<td>Yes</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>paycompensation_recurring</td>
<td>Yes</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>paycompensation_non_recurring</td>
<td>Yes</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>payment_information</td>
<td>Yes</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>job_relation</td>
<td>Yes</td>
<td>Yes</td>
<td></td>
</tr>
</tbody>
</table>
### Restriction of Fields

The restriction of fields in specific segments is not supported.

### 4.2.2 Time-Restricted Delta Transmission Support for MDF Objects

Delta transmission support for generic (MDF) objects in the CompoundEmployee API has timely restrictions.

The reason is that history information for MDF objects is only available after a given point in time and with delta transmission enablement of the MDF objects in the Q4 2014 or Q4 2015 release. Therefore, no historical data can be read and consequently no delta can be calculated if a point in time is chosen before the history data is available. Processing of the query request is terminated if an MDF object is part of the request using a date before these time restrictions.

**Note**

Affected MDF objects are: **deduction_recurring, deduction_non_recurring, alternative_cost_distribution**

Delta transmission support for the PaymentInformationV3 MDF object as well as the feature itself was introduced with the Q4 2014 release. Therefore, deltas only can be calculated from that release onwards. The IT Declaration MDF object was introduced in Q1 2015. Therefore, delta calculations are only supported from that release onwards.
4.2.3 WHERE Expression

In the WHERE expression, select parameters and operators can be used to restrict the query response.

Supported Select Parameters and Operators

The CompoundEmployee API doesn’t support expressions with complex conditions on multiple fields and different logical operators.

<table>
<thead>
<tr>
<th>Select Parameter</th>
<th>Remark</th>
<th>Operators</th>
</tr>
</thead>
<tbody>
<tr>
<td>LAST_MODIFIED_ON</td>
<td>Mandatory parameter. Returns all employees, for which any employee data has changed since this date and time. The date can only go back as far as 3 months.</td>
<td>&gt;</td>
</tr>
<tr>
<td>fromDate...toDate</td>
<td>Mandatory parameter for period-based delta transmission. Selects employees that have changes effective within the given period. Additionally, the period is applied as filter to all effective dated segments, so that only time slices are returned that intersect with the given period. This select parameter needs to be combined with the last_modified_on select parameter.</td>
<td>=</td>
</tr>
<tr>
<td>selectFromDate</td>
<td>Returns all employees that satisfy all job and/or compensation conditions where the effective end date of the respective time slice is greater than or equal to the value of selectFromDate. The value must be a fixed date format, for example: to_date('2016-01-01','yyyy-MM-dd')</td>
<td>=</td>
</tr>
<tr>
<td>selectToDate</td>
<td>Returns all employees that satisfy all job and/or compensation conditions where the effective start date of the respective time slice is lower than or equal to the value selectToDate. The value must be a fixed date format, for example: to_date('2016-01-01','yyyy-MM-dd')</td>
<td>=</td>
</tr>
<tr>
<td>Select Parameter</td>
<td>Remark</td>
<td>Operators</td>
</tr>
<tr>
<td>----------------------------------</td>
<td>------------------------------------------------------------------------------------------</td>
<td>-------------------</td>
</tr>
<tr>
<td>COMPANY_TERRITORY_CODE</td>
<td>Returns all employees that have a job at a company located in the provided country/region at any point in time. The companies are determined using table FO_LEGAL_ENTITY_T.</td>
<td>=, IN</td>
</tr>
<tr>
<td>PERSON_ID</td>
<td>Returns all employees with the respective PERSON_ID.</td>
<td>=, IN, NOT IN</td>
</tr>
<tr>
<td>PERSON_ID_EXTERNAL</td>
<td>Returns all employees with the respective PERSON_ID_EXTERNAL</td>
<td>=, IN, NOT IN</td>
</tr>
<tr>
<td>USER_ID</td>
<td>Returns all employees with the respective USER_ID</td>
<td>=, IN, NOT IN</td>
</tr>
<tr>
<td>COMPANY</td>
<td>Returns all employees that have a job at the selected company. The select is based on the external code of the company.</td>
<td>=, IN</td>
</tr>
<tr>
<td>EMPLOYEE_CLASS</td>
<td>Returns all employees that have a job of the respective EMPLOYEE_CLASS. The select is based on the external code of the employee class.</td>
<td>=, IN</td>
</tr>
<tr>
<td>DEPARTMENT</td>
<td>Returns all employees that have a job in the provided department. The select is based on the external code of the department.</td>
<td>=, IN</td>
</tr>
<tr>
<td>DIVISION</td>
<td>Returns all employees that have a job in the provided division. The select is based on the external code of the division.</td>
<td>=, IN</td>
</tr>
<tr>
<td>BUSINESS_UNIT</td>
<td>Returns all employees that have a job in the provided business unit. The select is based on the external code of the business unit.</td>
<td>=, IN</td>
</tr>
<tr>
<td>LOCATION</td>
<td>Returns all employees that have a job in the provided location. The select is based on the external code of the location.</td>
<td>=, IN</td>
</tr>
<tr>
<td>JOB_CODE</td>
<td>Returns all employees that have a job with the provided job code. The select is based on the external code of the job code.</td>
<td>=, IN</td>
</tr>
<tr>
<td>COMPENSATION_PAY_GROUP</td>
<td>Returns all employees that receive compensation with the provided pay group. The select is based on the external code of the pay group.</td>
<td>=, IN</td>
</tr>
<tr>
<td>Select Parameter</td>
<td>Remark</td>
<td>Operators</td>
</tr>
<tr>
<td>------------------</td>
<td>--------</td>
<td>-----------</td>
</tr>
<tr>
<td>SourceOfRecord</td>
<td>Returns all employees that have an employment with the provided source of record information. The select is based on the external code of the source of record information.</td>
<td>=, IN</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>isContingentWorker</td>
<td>Returns all employees that have an employment with the given value in field IsContingentWorker. The absence of the isContingentWorker condition is treated as isContingentWorker=false to preserve compatibility with older integration processes that expect real employees only. In order to select all persons that are contingent workers, use the condition isContingentWorker=true. If you want to request only regular employees; use the condition isContingentWorker=false. If you want both employment types, use condition isContingentWorker IN {true, false}.</td>
<td>=, IN</td>
</tr>
</tbody>
</table>

### i Note

The field needs to be enabled and made visible in the data model before this filter parameter can be used. Additionally, a picklist needs to be assigned to the Source Of Record field in the data model.

Possible comparison values for the IN operation and the equal operator are true, t, 1, yes, and false, f, 0, no. If the value in the field on the database is undefined (technically: null) it’s treated as false.

For more information about how to set up and configure Employee Central for contingent workers, refer to Contingent Workforce Management.
### Select Parameter

<table>
<thead>
<tr>
<th>Select Parameter</th>
<th>Remark</th>
<th>Operators</th>
</tr>
</thead>
<tbody>
<tr>
<td>hiringNotCompleted</td>
<td>Evaluates the indicator property hiringNotCompleted in the EmpEmployment entity in Employee Central, which was introduced for Onboarding. The property allows for differentiating data records of candidates (that is, new hires that didn’t yet complete the Manage Pending Hire process). If hiringNotCompleted is false, the CompoundEmployee API returns only data of hired employees. For more information about Onboarding, refer to Implementing Onboarding.</td>
<td>=</td>
</tr>
</tbody>
</table>

- **Note**
  Possible comparison values are: false, f, 0, no. That is, you can use the hiringNotCompleted filter in the WHERE condition only to exclude employments of candidates in the result.

### Effective Period Selection

Using the select parameters `selectFromDate` and `selectToDate`, you can further restrict the set of selected employees by specifying an open period, by using only one of the parameters, or by using both parameters to determine a fixed time range.

- **Note**
  You can only use effective period selection in combination with other select parameters of job_information or compensation_information. If you don’t use an additional parameter, the query returns an error.

### 4.2.4 Select Condition hiringNotCompleted

How to use the `hiringNotCompleted` select condition in the delta transmission mode of the CompoundEmployee API.

Use the `hiringNotCompleted` select condition in the query request if you want your downstream systems to receive only data that has been validated against the employee data model in Employee Central. The `hiringNotCompleted` parameter evaluates the indicator property `hiringNotCompleted` in the EmpEmployment entity in Employee Central, which was introduced for Onboarding. This property allows for differentiating data records of candidates (that is, new hires that didn’t yet complete the Manage Pending Hire process). For more information about Onboarding, refer to Implementing Onboarding.

The `hiringNotCompleted` field is part of the segment EMP_EMPLOYMENT_INFO. If the value for `hiringNotCompleted` of an employment is false, it’s a standard employment of a hired employee. If the value for `hiringNotCompleted` of an employment is true, it’s the employment of a candidate. That is, the employment is possibly incomplete with respect to the employee data model of Employee Central.

In contrast to other select conditions, such as company or employee_class, using `hiringNotCompleted` in the delta mode doesn’t lead to a CHANGE action when the corresponding field value changes. Employment
that had previously (at the last_modified_on timestamp) been filtered out according to the hiringNotCompleted condition require an INSERT action. Because only the INSERT action provides all data of the employment and all its requested subsegments. If the employment in question is the only employment of this person, also the person segment requires an INSERT action with all requested subsegments.

Let's look at some examples.

**Example (Initial Situation): One Single Employment Where hiringNotCompleted Is True**

**Snapshot**

<table>
<thead>
<tr>
<th>Person</th>
<th>Person1: PersA</th>
</tr>
</thead>
<tbody>
<tr>
<td>Employment</td>
<td>Employment1: EmplA (with hiringNotCompleted = true)</td>
</tr>
<tr>
<td>Job</td>
<td>Job1: JobA1</td>
</tr>
<tr>
<td></td>
<td>Job2: JobA2</td>
</tr>
</tbody>
</table>

**Current**

<table>
<thead>
<tr>
<th>Person</th>
<th>Person1: PersB</th>
</tr>
</thead>
<tbody>
<tr>
<td>Employment</td>
<td>Employment1: EmplB (with hiringNotCompleted = true)</td>
</tr>
<tr>
<td>Job</td>
<td>Job1: JobB1</td>
</tr>
<tr>
<td></td>
<td>Job2: JobB2</td>
</tr>
</tbody>
</table>

**Delta with** hiringNotCompleted = false

No result because the person is filtered out by the preselection

**Delta without** hiringNotCompleted condition

<table>
<thead>
<tr>
<th>Person</th>
<th>Change for Person1: PersA &gt; PersB</th>
</tr>
</thead>
<tbody>
<tr>
<td>Employment</td>
<td>Change for Employment1: EmplA &gt; EmplB</td>
</tr>
<tr>
<td>Job</td>
<td>Change for Job1: JobA1 &gt; JobB1</td>
</tr>
<tr>
<td></td>
<td>Change for Job2: JobA2 &gt; JobB2</td>
</tr>
</tbody>
</table>

**Note**

The CHANGE action can also be NO_CHANGE if the corresponding images of version A and B don’t differ.
Example (Transition Situation): One Single Employment Where hiringNotCompleted Changes from True to False

**Snapshot**

<table>
<thead>
<tr>
<th>Person</th>
<th>Person1: PersA</th>
</tr>
</thead>
<tbody>
<tr>
<td>Employment</td>
<td>Employment1: EmplA (with hiringNotCompleted = true)</td>
</tr>
</tbody>
</table>

**Current**

<table>
<thead>
<tr>
<th>Person</th>
<th>Person1: PersB</th>
</tr>
</thead>
<tbody>
<tr>
<td>Employment</td>
<td>Employment1: EmplB (with hiringNotCompleted = false)</td>
</tr>
</tbody>
</table>

**Delta with hiringNotCompleted = false**

<table>
<thead>
<tr>
<th>Person</th>
<th>Insert for Person1: PersB</th>
</tr>
</thead>
<tbody>
<tr>
<td>Employment</td>
<td>Insert for Employment1: EmplB (with hiringNotCompleted = false)</td>
</tr>
</tbody>
</table>

**Delta without hiringNotCompleted condition**

<table>
<thead>
<tr>
<th>Person</th>
<th>Change for Person1: PersA &gt; PersB</th>
</tr>
</thead>
<tbody>
<tr>
<td>Employment</td>
<td>Change for Employment1: EmplA (with hiringNotCompleted = true) &gt; EmplB (with hiringNotCompleted = false)</td>
</tr>
</tbody>
</table>

Example (Transition Situation): Multiple Employments Where hiringNotCompleted Changes from True to False for One Employment

**Snapshot**

<table>
<thead>
<tr>
<th>Person</th>
<th>Person1: PersA</th>
</tr>
</thead>
<tbody>
<tr>
<td>Employment</td>
<td>Employment1: EmplA1 (with hiringNotCompleted = false)</td>
</tr>
<tr>
<td></td>
<td>Employment2: EmplA2 (with hiringNotCompleted = true)</td>
</tr>
</tbody>
</table>

**Current**

<table>
<thead>
<tr>
<th>Person</th>
<th>Person1: PersB</th>
</tr>
</thead>
<tbody>
<tr>
<td>Employment</td>
<td>Employment1: EmplB1 (with hiringNotCompleted = false)</td>
</tr>
</tbody>
</table>

Employee Central Compound Employee API

CompoundEmployee API: Delta Transmission Mode
Employment2: EmplB2 (with hiringNotCompleted = false)

**Delta with hiringNotCompleted = false**

<table>
<thead>
<tr>
<th>Person</th>
<th>Change for Person1: PersA &gt; PersB</th>
</tr>
</thead>
<tbody>
<tr>
<td>Employment</td>
<td>Change for Employment1: EmplA1 &gt; EmplB1</td>
</tr>
<tr>
<td></td>
<td>Insert for Employment2: EmplB2 (with hiringNotCompleted = false)</td>
</tr>
</tbody>
</table>

**Delta without hiringNotCompleted condition**

<table>
<thead>
<tr>
<th>Person</th>
<th>Change for Person1: PersA &gt; PersB</th>
</tr>
</thead>
<tbody>
<tr>
<td>Employment</td>
<td>Change for Employment1: EmplA1 &gt; EmplB1</td>
</tr>
<tr>
<td></td>
<td>Change for Employment2: EmplA2 (with hiringNotCompleted = true) &gt; EmplB2 (with hiringNotCompleted = false)</td>
</tr>
</tbody>
</table>

**Special Handling When employment_information Segment Isn't Requested**

A special handling of the delta calculation is required when the employment_information segment isn't contained in the list of selected segments.

If the employment of a person has changed from hiringNotCompleted = true to hiringNotCompleted = false, an INSERT action must be done for the person if this is the only employment. If there’s another employment, which was replicated earlier, a CHANGE or NO_CHANGE action is required.

**Note**

If there’s no change in any of the requested segments, but the employment (that’s not requested) was changed, the result contains this person with the following log entry:

The employee has changed since the last replication on …, however, no data was returned because the changes are not relevant.

### 4.2.5 Sorting by Start Date in the WHERE Expression

Time slices of effective-dated objects in the response are by default sorted in descending order by start date.

This means that the newest time slice is always the first in the response. Semantic keys and sequence numbers are also considered in sorting.
The CompoundEmployee API supports ascending and descending sorting by start date for effective-dated entities. Add the following expression to the WHERE clause:

```
<queryString>
    SELECT person, personal_information, address_information,
    employment_information, job_information
    FROM CompoundEmployee
    ORDER BY start_date ASC / DESC
</queryString>
```

Table 14: Syntax of ORDER BY Expressions

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Sorting Order of Effective-Dated Entities</th>
</tr>
</thead>
<tbody>
<tr>
<td>ORDER BY start_date DESC</td>
<td>descending</td>
</tr>
<tr>
<td>ORDER BY start_date ASC</td>
<td>ascending</td>
</tr>
<tr>
<td>ORDER BY start_date</td>
<td>ascending</td>
</tr>
</tbody>
</table>

4.2.6 Controlling Parameters for Delta Transmission

You can use these parameters to control the delta transmission mode of the CompoundEmployee API.

Delta transmission needs be enabled and requires always a date/time indication for the last synchronization as part of the WHERE clause when calling the CompoundEmployee API.

```
FROM CompoundEmployee
WHERE ( last_modified_on > to_datetime(LastModifiedDate) 
```

For both delta transmission modes, the query parameter queryMode and value delta or periodDelta needs to be passed to run the API in delta transmission mode.

```
<urn:param>
    <urn:name>queryMode</urn:name>
    <urn:value>delta</urn:value>
</urn:param>
```

```
<urn:param>
    <urn:name>queryMode</urn:name>
    <urn:value>periodDelta</urn:value>
</urn:param>
```

To indicate the period for period-based delta transmission, the parameter couple fromDate ... toDate is required as part of the WHERE clause together with the last_modified_on date when calling the API. This
period parameter couple needs to be passed with an AND operator in the WHERE clause. From selection point of view, employees need to match either the last_modified_on or the period criterion in order to get selected.

```
FROM CompoundEmployee
WHERE ( last_modified_on > to_datetime(LastModifiedDate)
    AND fromDate = to_date(FromDate,'YYYY-MM-DD')
    AND toDate   = to_date(ToDate,'YYYY-MM-DD') )
```

### 4.2.6.1 Parameter changedSegmentsOnly

Consumers who are only interested in changed segments can call the CompoundEmployee API delta mode with the parameter changedSegmentsOnly.

Delta transmission can be used in two different modes. In standard mode, the complete history for an employee for all requested segments is returned. Such a query response XML includes action code information to indicate which data has changed and for which information no change has happened. The advantage is that the consumer receives complete information about all employee records when using effective-dated delta transmission, but receives changes indicated from the action code. This is useful in situations where the consumer must process information although it hasn’t changed in Employee Central.

#### Example

The change of a field in the job_information segment (for example, change of legal entity, country/region, or pay group or the termination and rehire of the employee) can make it necessary to re-create the employee with all their data. In such a case, it’s beneficial to the consumer that person-specific data such as address or name is sent with the query response even though this data hasn’t changed.

#### Note

For period-based delta transmission, only the unchanged segments are returned that intersect with the given period.

To request changed segments only, call the delta mode with the parameter changedSegmentsOnly:

```
<urn:param>
    <urn:name>resultOptions</urn:name>
    <urn:value>changedSegmentsOnly</urn:value>
</urn:param>
```

If this parameter is passed using the query string, the CompoundEmployee API filters out the segments with the action code NO CHANGE, so that only INSERT, CHANGE, and DELETE information is communicated. However, the hierarchical structure is considered. In case there’s a change on a subsegment, the unchanged super-segment is returned as well.

```
<employment_information>
    <action>NO CHANGE</action>
</employment_information>
```
4.2.6.2 Parameter Value isNotFirstQuery

Use the isNotFirstQuery parameter value to avoid duplicates in the delta transmission mode of the CompoundEmployee API.

The period-based delta transmission can be used in two different ways. The standard is to get information about employees with effective-dated objects effective during the given period. In this case, the effective-dated objects are replicated with the action code INSERT or CHANGE since the CompoundEmployee API assumes that they aren’t yet replicated. To avoid duplicates and to identify real changes happening within the period, the CompoundEmployee API can be executed applying the resultOptions parameter with the value isNotFirstQuery, for example, a scenario where multiple replications happen in the same payroll period.

Example

Period-based delta transmission can be used to replicate data for such a payroll period use case, where an extraction of data is triggered multiple times. In this scenario, data is replicated once it becomes effective for the period, for example, a new salary. Changes of the data are replicated in subsequent runs where only corrections in the same period to the payroll system are sent, such as corrections of accidentally entered amounts for wage increases.

Sample Code

```xml
<urn:param>
  <urn:name>resultOptions</urn:name>
  <urn:value>isNotFirstQuery</urn:value>
</urn:param>
```

If the parameter value isNotFirstQuery is used, the CompoundEmployee API assumes the period matching effective-dated objects as already replicated and calculates deltas between current and snapshot image. Action codes are therefore also INSERT, CHANGE, or DELETE, depending on the state of data that is already replicated.
4.3 Delta Transmission Query Response

Here’s what the query response of the CompoundEmployee API call looks like.

The query response has the following properties:

- **sfoObject**: List of employees with subentities, returned as hierarchical XML
- **numResult**: Number of top-level results
- **hasMore**: Indicator if more results are available
- **querySessionId**: Query session ID for paging

The query response follows the current Employee Central data model by building a hierarchically structured XML. The names of the substructures are derived from the underlying database tables and SAP SuccessFactors entity names. The leaf elements below are named according to HRIS field metadata (HRIS-field ID or HRIS-field name) or MDF object instance name.

The CompoundEmployee API returns employees (persons with employment) only. Persons without employment are excluded from extraction and not returned. This means that technical users and employee dependents aren’t returned as sfoObject by the API. But dependents are considered as a subelement of employees.

4.3.1 Query Response for Effective-Dated and Period-Based Delta Transmission

The query response of the CompoundEmployee API has some specifics regarding effective-dated and period-based delta transmission.

Effective-Dated Delta Transmission

In the effective-dated delta transmission mode, the CompoundEmployee API selects all employees that were changed since the last synchronization, determines the snapshot image and the current image, and compares both images to calculate the delta. The response contains the calculated changes independent on whether objects in the past or if future dated objects were affected. Therefore, the consumer needs to be able to deal with effective dated objects and their time slices.
Period-Based Delta Transmission

In period-based delta transmission mode, the CompoundEmployee API considers not only retroactive changes but also employees for which changes become effective in the provided period. But changes are ignored that become effective after the period.

In period-based delta transmission, the following criteria are considered to determine relevant employees for the selection:

- Time slices starting within the provided period (like the third Personal Info time slice in figure) or
- Changed time slices starting before the period (like the second Job Info time slice in figure) or
- Changes of not effective dated entities

For the selected employees, the CompoundEmployee API returns the following relevant data
● All time slices that intersect with the provided period (see the second and the third Personal Info time slice and the third Job Info time slice in figure)
● All time slices that start before the provided period and were changed (see the second Job Info time slice in figure)
● All changed non-effective-dated entities

Figure 10: Data of Employees Replicated When Using Period-Based Delta Transmission

Non-Effective-Dated Objects

Non-effective-dated objects such as national IDs or email are returned in both kinds of delta transmissions, if they're changed after the given last_modified_on timestamp for the last replication.

In the example, a delta transmission response of the CompoundEmployee API is shown, in which an employee is returned who married on June 1, 2014.

Sample Code

```xml
<person>
  <action>NO CHANGE</action>
  <created_by>root</created_by>
  <created_on_timestamp>2014-05-14T07:20:55.000Z</created_on_timestamp>
  <last_modified_by>root</last_modified_by>
  <last_modified_on>2014-05-14T07:20:55.000Z</last_modified_on>
  <last_saved_on>2014-05-14T07:21:02.793Z</last_saved_on>
  <logon_user_id>261</logon_user_id>
  <logon_user_is_active>true</logon_user_is_active>
  <logon_user_name>261</logon_user_name>
  <person_id>1180</person_id>
  <person_id_external>261</person_id_external>
  <personal_information>
    <action>CHANGE</action>
    <created_by>root</created_by>
    <created_on_timestamp>2014-05-14T07:20:56.000Z</created_on_timestamp>
    <last_modified_by>root</last_modified_by>
    <last_modified_on>2014-05-14T07:20:55.000Z</last_modified_on>
    <last_saved_on>2014-05-14T07:21:02.793Z</last_saved_on>
    <marital_status>M</marital_status>
    <previous>S</previous>
    <start_date>2014-05-01</start_date>
  </personal_information>
</person>
```
Highlighted fields are:

- The action code for the not effective-dated person segment, which isn’t changed. Therefore, it’s set to **NO CHANGE**.
- The action code for the effective-dated personal information segment, which is set to **CHANGE** because of the change in marital status from single (S) to married (M).
- The marital status showing the new value (M) and the previous value (S).

### 4.4 Action Codes

The delta transmission mode of the CompoundEmployee API supports some action codes.

#### Supported Action Codes

<table>
<thead>
<tr>
<th>Action Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>INSERT</td>
<td><strong>INSERT</strong> is communicated if a new record is created. For example, when an employee is hired, all segments are returned with action code <strong>INSERT</strong>.</td>
</tr>
<tr>
<td>CHANGE</td>
<td><strong>CHANGE</strong> is sent if an existing record has been modified. For example, the amount of a bonus has been adjusted.</td>
</tr>
<tr>
<td>DELETE</td>
<td><strong>DELETE</strong> means that a record has been deleted completely. For example, paycompensation_non_recurring information has been removed.</td>
</tr>
<tr>
<td>NO_CHANGE</td>
<td><strong>NO_CHANGE</strong> indicates that there were no relevant changes on the exposed attributes of the segment since the last replication. In case a segment is effective dated, several instances of this segment might be in the response and each can have a different action code. In cases where one instance has the action code <strong>NO_CHANGE</strong>, this means that in the time frame of this instance no differences exist compared to the data status at the last replication.</td>
</tr>
</tbody>
</table>

The action codes are applied to effective-dated segments (like **personal_information**) as well as to non-effective-dated entities (such as **national_id_card**). In case of effective-dated segments, the consumer should be aware that the action code doesn’t always resemble the action that has been performed on the UI.

#### Example: Action Code CHANGE

An employee changes their job title on June 1. This means that a second job slice is created starting from June 1 and that the first job slice is delimited. In this case, the second job slice isn’t communicated with the action code **INSERT**, but with the action code **CHANGE**. The reason is that job data for the employee existed since the
hire date of the employee. The second job slice only replaces the data records of the original hire job slice. Therefore, it’s a CHANGE.

4.5 Previous Fields

In case a field has been changed since the last modified date, a subelement called previous is added to the field by the CompoundEmployee API.

While the field itself contains the current value of the field, the previous element shows which content the field had before the change.

Three data constellations are possible leading to three different types of API responses:

### No Information Existed Before

Before the last modified date, no information existed about the employee’s marital status. In the meantime, the marital status was changed to *Single* (S).

```xml
<marital_status>
  S
  <previous/>
</marital_status>
```

### Information Is Changed

Before the last modified date, the employee’s marital status was *Single* (S). In the meantime, the marital status was changed to *Married* (M).

```xml
<marital_status>
  M
  <previous>S</previous>
</marital_status>
```

### Information Is Deleted

Before the last modified date, the employee’s marital status was *Married* (M). In the meantime, the marital status information was deleted.
4.6 Event Information

Job information and compensation information require special treatment because they contain the information about the business events, for example, a newly-hired employee.

Whenever a new slice of job information or compensation information is created, this is triggered by a certain business event. Therefore, each slice of job and compensation contains event information.

Typical business events in Employee Central include:

- Hire
- Promotion
- Leave
- Job change
- Termination
- Rehire

Many of the business events are also important events for the consumer. Based on the events, different processes might be triggered on the consumer side. Also different exchange formats such as HR-XML position the business event as a central entity.

Therefore, in delta mode, the business events contained in job and confirmation slices are transferred in their own segments. They are non-effective-dated but have an event date that corresponds to the start date of the underlying job or compensation time slice.

It is possible that several events happen on the same date. In this case, several concurrent time slices are created that differ in their sequence number. The time slices that do not have the highest sequence number do not cover the whole time period but have an end date that equals to their start date. In the example below, the second time slice has three records with different business events. Each record has a transaction sequence number. The record with the highest sequence number contains the business data for the time slice. That’s the reason why, in standard delta mode, only the job data or compensation data of the time slice with the highest sequence number is considered.

<table>
<thead>
<tr>
<th>A (Seq = 1, Event = Hire)</th>
<th>B (Seq = 1, Event = Promotion)</th>
</tr>
</thead>
<tbody>
<tr>
<td>C (Seq = 2, Event = Data Change)</td>
<td></td>
</tr>
<tr>
<td>D (Seq = 3, Event = Leave)</td>
<td></td>
</tr>
</tbody>
</table>

Figure 11: Example of Time Slices

The next sections describe the event handling in detail on basis of job information. Note that the handling of the events of compensation information is mainly the same.
4.6.1 Insert New Job Time Slice with Multiple Events

This is the event handling when multiple time slices are inserted with the same start date and different events. The delta processor calculates the delta for job information on basis of the records with the highest sequence number for each time slice. The job events are calculated for each start date by comparing the events valid on this start date in the snapshot data and the current data.

**Snapshot Data**

| A (Seq = 1, Event = Hire) |

Figure 12: Snapshot

**Current Data**

| A (Seq = 1, Event = Hire) | B (Seq = 1, Event = Promotion) | C (Seq = 2, Event = Data Change) | D (Seq = 3, Event = Leave) |

Figure 13: Current

**Delta to Be Transferred: Job**

| A (Seq = 1, Event = Hire) | Charge D (A -> D) (Seq = 3, Event = Leave) |

Figure 14: Job

**Delta to Be Transferred: Job**

| Hire | Insert Promotion | Insert Data Change | Insert Leave |

Figure 15: Job Events
4.6.2 Insert New Job Time Slice to Existing Time Slice

This is the event handling when multiple time slices are with the same start date and different events are added to an already existing slice with the same start date.

Snapshot Data

| A (Seq = 1, Event = Hire) | B (Seq = 1, Event = Promotion) |

Figure 16: Snapshot

Current Data

<table>
<thead>
<tr>
<th>A (Seq = 1, Event = Hire)</th>
<th>B (Seq = 1, Event = Promotion)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>C (Seq = 2, Event = Data Change)</td>
</tr>
<tr>
<td></td>
<td>D (Seq = 2, Event = Leave)</td>
</tr>
</tbody>
</table>

Figure 17: Current

Delta to Be Transferred: Job

| A (Seq = 1, Event = Hire) | Change D (B→D) (Seq = 2, Event = Leave) |

Figure 18: Job

Delta to Be Transferred: Job Events

<table>
<thead>
<tr>
<th>Hire</th>
<th>Promotion</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Insert Data Change</td>
</tr>
<tr>
<td></td>
<td>Insert Leave</td>
</tr>
</tbody>
</table>

Figure 19: Job Events
4.6.3 Change Event of a Concurrent Time Slice

This is the event handling when an event in one of the concurrent time slices is changed.

Snapshot Data

<table>
<thead>
<tr>
<th>A (\text{Seq} = 1, \text{Event} = \text{Hire})</th>
<th>B (\text{Seq} = 1, \text{Event} = \text{Promotion})</th>
</tr>
</thead>
<tbody>
<tr>
<td>C (\text{Seq} = 2, \text{Event} = \text{Leave})</td>
<td>D (\text{Seq} = 3, \text{Event} = \text{Data Change})</td>
</tr>
</tbody>
</table>

Figure 20: Snapshot

Current Data

<table>
<thead>
<tr>
<th>A (\text{Seq} = 1, \text{Event} = \text{Hire})</th>
<th>B (\text{Seq} = 1, \text{Event} = \text{Promotion})</th>
</tr>
</thead>
<tbody>
<tr>
<td>C (\text{Seq} = 2, \text{Event} = \text{Job Change})</td>
<td>D (\text{Seq} = 3, \text{Event} = \text{Data Change})</td>
</tr>
</tbody>
</table>

Figure 21: Current

Delta to Be Transferred: Job

| A \(\text{Seq} = 1, \text{Event} = \text{Hire}\) | D \(\text{Seq} = 3, \text{Event} = \text{Data Change}\) |

Figure 22: Job

Delta to Be Transferred: Job Events

<table>
<thead>
<tr>
<th>Hire</th>
<th>Promotion</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Delete Leave</td>
</tr>
<tr>
<td></td>
<td>Insert Job Change</td>
</tr>
<tr>
<td></td>
<td>Data Change</td>
</tr>
</tbody>
</table>

Figure 23: Job Events

On the UI, this might be a simple change of job field. However, from a business perspective, it means that the employee does not go on leave. Therefore, an explicit delete must be sent for the Leave event.

The job data change is not communicated because the Compound API only sends job data of the concurrent job slice with the highest sequence number.
4.6.4 Change Not Event Related Job Field of a Concurrent Time Slice

This is the event handling when a not event-related field in one of the concurrent time slices is changed.

Snapshot Data

<table>
<thead>
<tr>
<th>A (Seq = 1, Event = Hire)</th>
<th>B (Seq = 1, Event = Promotion, Department = C)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>C (Seq = 2, Event = Data Change, Department = A)</td>
</tr>
<tr>
<td></td>
<td>D (Seq = 3, Event = Leave, Department = C)</td>
</tr>
</tbody>
</table>

Figure 24: Snapshot

Current Data

<table>
<thead>
<tr>
<th>A (Seq = 1, Event = Hire)</th>
<th>B (Seq = 1, Event = Promotion, Department = C)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>C (Seq = 2, Event = Data Change, Department = B)</td>
</tr>
<tr>
<td></td>
<td>D (Seq = 3, Event = Leave, Department = C)</td>
</tr>
</tbody>
</table>

Figure 25: Current

Delta to Be Transferred: Job

| A (Seq = 1, Event = Hire) | D (Seq = 3, Event = Leave) |

Figure 26: Job

Delta to Be Transferred: Job Events

<table>
<thead>
<tr>
<th>Hire</th>
<th>Promotion</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Data Change</td>
</tr>
<tr>
<td></td>
<td>Leave</td>
</tr>
</tbody>
</table>

Figure 27: Job Events

As the event has not been changed, all events are sent with action NO CHANGE. The job data change on C is not relevant in standard delta mode since only the job slice with the highest sequence number is considered.
4.6.5 Delete Job Time Slice (Active)

This is the event handling when the record with the highest transaction sequence number of a time slice is deleted.

Snapshot Data

<table>
<thead>
<tr>
<th>A (Seq = 1, Event = Hire)</th>
<th>B (Seq = 1, Event = Promotion)</th>
</tr>
</thead>
<tbody>
<tr>
<td>C (Seq = 2, Event = Data Change)</td>
<td></td>
</tr>
<tr>
<td>D (Seq = 3, Event = Leave)</td>
<td></td>
</tr>
</tbody>
</table>

Figure 28: Snapshot

Current Data

<table>
<thead>
<tr>
<th>A (Seq = 1, Event = Hire)</th>
<th>B (Seq = 1, Event = Promotion)</th>
</tr>
</thead>
<tbody>
<tr>
<td>C (Seq = 2, Event = Data Change)</td>
<td></td>
</tr>
</tbody>
</table>

Figure 29: Current

Delta to Be Transferred: Job

| A (Seq = 1, Event = Hire) | Change C (D→C, Seq 3 → 2) (Seq = 2, Event = Data Change) |

Figure 30: Job

The delta information indicates that the job data of time slice C is valid now.

Delta to Be Transferred: Job Events

Hire

| Promotion |
| Data Change |
| Delete Leave |

Figure 31: Job Events
4.6.6 Delete Job Time Slice (In Between)

This is the event handling when the record of a time slice is deleted which does not have the highest transaction sequence number.

Snapshot Data

<table>
<thead>
<tr>
<th>A (Seq = 1, Event = Hire)</th>
<th>B (Seq = 1, Event = Promotion)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>C (Seq = 2, Event = Data Change)</td>
</tr>
<tr>
<td></td>
<td>D (Seq = 3, Event = Leave)</td>
</tr>
</tbody>
</table>

Figure 32: Snapshot

Current Data

<table>
<thead>
<tr>
<th>A (Seq = 1, Event = Hire)</th>
<th>B (Seq = 1, Event = Promotion)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>D (Seq = 2, Event = Leave)</td>
</tr>
</tbody>
</table>

Figure 33: Current

Delta to Be Transferred: Job

| A (Seq = 1, Event = Hire) | Change D (Seq 3 → 2) (Seq = 2, Event = Leave) |

Figure 34: Job

Delta to Be Transferred: Job Events

<table>
<thead>
<tr>
<th>Hire</th>
<th>Promotion</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Delete Data Change</td>
</tr>
<tr>
<td></td>
<td>Leave</td>
</tr>
</tbody>
</table>

Figure 35: Job Events
4.7 Effective-Dated and Period-Based Delta Transmission

The CompoundEmployee API offers two different kinds of delta transmission: effective-dated and period-based.

Prerequisites

Delta transmission is based on an active auditing to determine the historical data set of employees. This means that auditing must be switched on during the system setup phase before the initial load of employees into the system.

Effective-Dated Delta Transmission

Effective-dated delta transmission is designed for consumers that work in an effective-dated manner. This means, it’s 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

Period-based delta transmission is intended for consumers that aren’t able to deal with effective-dated objects. Deltas contain retroactive changes and effective-dated objects that are relevant for the given period. Non-effective-dated objects are returned in both kinds of delta transmissions if they’re changed after the last replication.

Example

Let’s say an employee was hired and created in the system in September. In mid-October, the job data of the employee was changed using Make correction. Today (end of October), the salary was changed effective mid-October and the address was changed becoming effective in November. The last synchronization of the consumer took place before the change of the job data.
In effective-dated delta transmission, the CompoundEmployee API returns all changes that happened since the last synchronization, such as the change of the job, the change of the salary, and the future address change.

For a consumer using period-based delta transmission and providing October 1 to October 31 as synchronization period, only the change of the job and the change of the salary will be replicated since these changes affect the given period and the data that has already been replicated to the consumer in the past. However, the change of the address remains unconsidered and will be transmitted in the next period of November, when it becomes effective.

4.7.1 Effective-Dated Delta Transmission with the CompoundEmployee API

Effective-dated delta transmission considers all changes that have happened from a given point in time.

The CompoundEmployee API query response contains all changes since that point in time happened for non-effective-dated and effective-dated entities including retroactive and future dated changes. Therefore, this type of delta transmission is intended for consumers that are able to handle time slices and future dated information.
4.7.1.1 Use Cases for Effective-Dated Entities

We differentiate time slice handling between effective-dated entities without semantic key (such as job information) and effective-dated entities with semantic key (such as address information).

4.7.1.1.1 Time Slice Handling for Effective-Dated Entities Without Semantic Key

Learn more about how the CompoundEmployee API handles time slices for changes to effective-dated entities without semantic key, such as job information or compensation.

Changes to effective-dated entities without semantic key, such as Job Information or Compensation, are always calculated against that historical state of the snapshot data of the employee.

4.7.1.1.1.1 Insertion of a Time Slice

This is the handling when a new time slice is inserted.

First Time Slice Is Added

Figure 37: Snapshot

Figure 38: Current

Figure 39: Delta to Be Transferred

The first time slice A is added, the snapshot data does not contain any time slice. The delta contains the time slice A with action INSERT.
Second Time Slice Is Added After the First Time Slice

The delta contains two time slices. The first time slice has the same values as before, but its end date is delimited to the start date of the new inserted time slice. Its action code is \texttt{NO\_CHANGE} since no values were changed. The second time slice has action code \texttt{CHANGE} because the values have changed from A to B in this time period. The \texttt{CHANGE} is returned because the object already existed and could only be changed.

Third Time Slice Is Added After the Second Time Slice

The delta contains three time slices: The first time slice remains completely unchanged. The second time slice has the same values as before, only its end date is delimited to the start date of the new inserted time slice. For the first and the second time slice, the action code is \texttt{NO\_CHANGE}, since no values were changed. Only the third time slice has the action code \texttt{CHANGE} because the value changed from B to C in this time period.

New Time Slice Is Inserted Before the First Time Slice
The delta contains three time slices. The first time slice is the inserted time slice with values C and has action code **INSERT**. The other time slices are not changed and therefore have action code **NO_CHANGE**.

### New Time Slice Is Inserted After the First Time Slice

The delta contains three time slices. The first time slice has the same values as before, but its end date is delimited to the start date of the new inserted time slice. Its action code is **NO_CHANGE** since no values were changed. The same applies to the third time slice, which remains completely unchanged. Only the second time slice has action code **CHANGE** because the values have changed from A to C in this time period.
4.7.1.1.2 Change of Values

This is the handling when some values of an existing time slice were changed using Make Correction.

Variant 1: Single Time Slice Is Changed

Snapshot

| A |

Current

| B |

Delta to be transferred

| Change B (A→B) |

Figure 52: Changing a Single Time Slice

The delta contains the first time slice with action code CHANGE and the new value B.

Variant 2: One of Several Time Slices Is Changed

Snapshot

| A | B |

Current

| C | B |

Delta to be transferred

| Change C (A→C) | B |

Figure 53: Changing One of Multiple Time Slices

The delta contains the first time slice with action code CHANGE and the new value C. The second time slice has action code NO_CHANGE and the unchanged value B.
4.7.1.1.1.3 Deletion of a Time Slice

This is the handling when a time slice is deleted.

**Variant 1: First Time Slice Is Deleted**

<table>
<thead>
<tr>
<th>Snapshot</th>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>Current</td>
<td></td>
<td>B</td>
</tr>
<tr>
<td>Delta to be transferred</td>
<td>Delete A</td>
<td>B</td>
</tr>
</tbody>
</table>

*Figure 54: Deletion of First Time Slice*

The deletion of the first time slice is transmitted as an explicit deletion with action code DELETE. The second time slice is not affected by the change and has action code NO CHANGE.

**Variant 2: Time Slice After First Time Slice Is Deleted**

<table>
<thead>
<tr>
<th>Snapshot</th>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>Current</td>
<td>A</td>
<td></td>
<td>C</td>
</tr>
<tr>
<td>Delta to be transferred</td>
<td>A</td>
<td>Change A (B→A)</td>
<td>C</td>
</tr>
</tbody>
</table>

*Figure 55: Deletion of Second Time Slice*

The delta contains three time slices. The first time slice has the same values as before but its end date is delimited to the start date of the deleted time slice. Its action code is NO_CHANGE since no values were changed. The same applies to the third time slice which remains completely unchanged. The second time slice has the same values as the first time slice but compared to the snapshot data the value changed from B to A. Therefore the second time slice has the action code CHANGE.
4.7.1.1.4 Move of Start Date

This is the handling when a time slice is moved to an earlier or a later point in time.

**Move Start Date to Earlier Point in Time**

**Variant 1: The start date of a time slice is modified to an earlier point in time and the values of the time slice have not changed**

<table>
<thead>
<tr>
<th>Snapshot</th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>B</td>
<td>C</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Current</th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>B</td>
<td>C</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Delta to be transferred</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>Change B (A→B)</td>
</tr>
</tbody>
</table>

Figure 56: Earlier Start Date Without Data Changes

In the delta, the changed time slice is split into two time slices – one time slice for the period where the values have changed and one time slice for the period without changes.

**Variant 2: The start date of a time slice is modified to an earlier point in time and the values of the time slice have changed**

<table>
<thead>
<tr>
<th>Snapshot</th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>B</td>
<td>C</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Current</th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>D</td>
<td>C</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Delta to be transferred</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>Change D (A→D)</td>
</tr>
</tbody>
</table>

Figure 57: Earlier Start Date With Data Changes

Also in this case, the changed time slice is split into two time slices, but now both time slices have the action code CHANGE since the values have been changed in both time periods.
Move Start Date to Later Point in Time

Variant 1: The start date of the first time slice is modified to a later point in time and values have changed

<table>
<thead>
<tr>
<th>Snapshot</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>A</td>
<td>B</td>
</tr>
<tr>
<td>Current</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Delta to be transferred</td>
<td>Delete A</td>
<td>Change C [A → C]</td>
</tr>
</tbody>
</table>

Figure 58: Later Start Date of First Time Slice, With Data Changes

The delta contains three time slices. The first time slice has the start date of the original first time slice of the snapshot data and is delimited to the new start date of the changed time slice. Its action code is DELETE since the time period is no longer covered in the current data. The second time slice of the delta corresponds to the first time slice of the current data and has CHANGE as action code since the value changed from A to C. The third time slice is unchanged and has action code NO_CHANGE.

Variant 2: The start date of a time slice after the first time slice is modified to a later point in time and the values of the time slice have changed

<table>
<thead>
<tr>
<th>Snapshot</th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>A</td>
<td>B</td>
<td>C</td>
</tr>
<tr>
<td>Current</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>A</td>
<td>D</td>
<td>C</td>
</tr>
</tbody>
</table>

Figure 59: Later Start Date of Second Time Slice, With Data Changes

The delta contains four time slices. The first time slice has the same values as before, but its end date is delimited to the start date of the second time slice of the snapshot data. Its action code is NO_CHANGE since no values were changed. The same applies to the last time slice, which remains unchanged. The second time slice has the same values as the first time slice, but compared to the snapshot data the value changed from B to A. Therefore the second time slice has the action code CHANGE. The third time slice of the delta corresponds to the second time slice of the current data and has previous value B.
4.7.1.2 Time Slice Handling for Effective-Dated Entities with Semantic Key

Learn more about how the CompoundEmployee API handles time slices for changes to effective-dated entities with semantic key, such as Address Information.

The main difference compared to the effective-dated entities without semantic key is that each time slice can have several records with different semantic keys. For Address Information, for example, the semantic key is defined by the address type and country/region. A time slice can have multiple addresses of different types:

![Figure 60: Example of Address Types](image)

For such entities, we calculate the deltas for each semantic key and time slice.

The following entities are effective-dated with semantic key:

Table 15: Effective-Dated Entities with Semantic Key

<table>
<thead>
<tr>
<th>Entity</th>
<th>Key</th>
</tr>
</thead>
<tbody>
<tr>
<td>Person Global Info</td>
<td>country</td>
</tr>
<tr>
<td>Home Address</td>
<td>address_type</td>
</tr>
<tr>
<td></td>
<td>country</td>
</tr>
<tr>
<td>Person Relationship</td>
<td>relationship_type</td>
</tr>
<tr>
<td></td>
<td>related_person_id</td>
</tr>
<tr>
<td>Pay Component Recurring</td>
<td>pay_component</td>
</tr>
<tr>
<td>Job Relation</td>
<td>relationship_type</td>
</tr>
</tbody>
</table>
4.71.2.1 Change of Values

This is the handling when some values of an existing time slice were changed.

Change a Standard Attribute of the Address of Type Vacation

<table>
<thead>
<tr>
<th>Snapshot</th>
<th>Home</th>
<th>HA</th>
<th>VB</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vacation</td>
<td>VA</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Current</th>
</tr>
</thead>
<tbody>
<tr>
<td>Home</td>
</tr>
<tr>
<td>Vacation</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Delta to be transferred</th>
</tr>
</thead>
<tbody>
<tr>
<td>Home</td>
</tr>
<tr>
<td>Vacation</td>
</tr>
</tbody>
</table>

Figure 61: Changing a Time Slice

In this example, the home address hasn’t changed and therefore all records with this semantic key have the action code NO CHANGE. The vacation address was changed from VA to VB and therefore has the action code CHANGE.

Insert a New Time Slice with a Different Country/Region

A new time slice with a different country/region is added for the address of address type Home.

<table>
<thead>
<tr>
<th>Snapshot</th>
</tr>
</thead>
<tbody>
<tr>
<td>Home</td>
</tr>
<tr>
<td>Vacation</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Current</th>
</tr>
</thead>
<tbody>
<tr>
<td>Home</td>
</tr>
<tr>
<td>Vacation</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Delta to be transferred</th>
</tr>
</thead>
<tbody>
<tr>
<td>Home</td>
</tr>
<tr>
<td>Vacation</td>
</tr>
</tbody>
</table>

Figure 62: Inserting a Time Slice

In this example, a new time slice was inserted for the vacation address and the home address. Since the attributes of the vacation address weren’t changed in the new time slice, all records of the vacation address
have the action code NO CHANGE. For the home address, the address data and specifically the country/region were changed in the new time slice. Since the country/region is part of the semantic key of the address, the delta calculation returns a delete of the previous address in the United States and an insert of the new address in Germany for the new time slice.

**Insert a New Time Slice and Remove a Record**

A new time slice is inserted, the record of the vacation address is removed, and the home address is changed in the new time slice.

<table>
<thead>
<tr>
<th>Snapshot</th>
<th>Current</th>
<th>Delta to be transferred</th>
</tr>
</thead>
<tbody>
<tr>
<td>Home</td>
<td>HA</td>
<td>HA</td>
</tr>
<tr>
<td>Vacation</td>
<td>VA</td>
<td>HA</td>
</tr>
<tr>
<td></td>
<td></td>
<td>HC</td>
</tr>
<tr>
<td></td>
<td></td>
<td>VA</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Delete VA</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Change HC (HA→HC)</td>
</tr>
</tbody>
</table>

In this case, the delta contains three time slices and each time slice has a record for the home address and a record for the vacation address. In the first and third time slice, all records have the action code NO CHANGE. In the second time slice, the record of the home address has the action CHANGE since the value changed from HA to HC. The record of the vacation address has the action code DELETE since it was deleted in this time period.

**4.7.1.3 Example: Compensation and Pay Compensation Recurring**

Here's a complex example of delta calculation for compensation and pay compensation recurring.

In the example, the time slices of the snapshot data and the current data are completely different, which leads to a split of time slices in the delta.
4.7.1.4 Non-Effective-Dated Entities

Look at delta transmission for non-effective-dated entities such as paycompensation_non_recurring, direct_deposit, or national_id_card.

For such entities, the delta processor checks whether a record was inserted, changed, or deleted and extracts it with the corresponding action code. The records are identified by a semantic key, which is defined for each entity:

Table 16: Semantic Keys for Non-Effective-Dated Entities

<table>
<thead>
<tr>
<th>Entity</th>
<th>Key</th>
</tr>
</thead>
<tbody>
<tr>
<td>direct_deposit</td>
<td>deposit_type</td>
</tr>
<tr>
<td></td>
<td>routing_number</td>
</tr>
<tr>
<td></td>
<td>account_number</td>
</tr>
<tr>
<td></td>
<td>account_type</td>
</tr>
<tr>
<td></td>
<td>process_type</td>
</tr>
<tr>
<td>national_id_card</td>
<td>country</td>
</tr>
<tr>
<td></td>
<td>card_type</td>
</tr>
</tbody>
</table>
Changing attributes that are part of the semantic key always leads to the extraction of two records: One record with the action DELETE and the old semantic key and one record with the action code INSERT and the new semantic key. For example, if the pay date of a non-recurring pay compensation is changed, the delta contains a record with the action code DELETE with the old pay date and a record with the action code INSERT and the new pay date.

Example: Delta Calculation for Change of Pay Compensation Non-Recurring

<table>
<thead>
<tr>
<th>Entity</th>
<th>Key</th>
</tr>
</thead>
<tbody>
<tr>
<td>paycompensation_non_recurring</td>
<td>pay_component_code</td>
</tr>
<tr>
<td></td>
<td>pay_date</td>
</tr>
<tr>
<td></td>
<td>sequence_number</td>
</tr>
<tr>
<td>email_information</td>
<td>email_type</td>
</tr>
<tr>
<td>phone_information</td>
<td>phone_type</td>
</tr>
</tbody>
</table>

**Before Data**

| Spot Bonus | 31-Mar-2014 | 1000€ |

**Current Data**

| Spot Bonus | 31-Mar-2014 | 1500€ |

**Delta**

| Change | Spot Bonus | 31-Mar-2014 | 1500€ |

Figure 65: Changing Pay Compensation Non-Recurring
Example: Delta Calculation for Change of Key Field of Pay Compensation Non-Recurring

<table>
<thead>
<tr>
<th>Before Data</th>
<th>Spot Bonus</th>
<th>31-Mar-2014</th>
<th>1500€</th>
</tr>
</thead>
<tbody>
<tr>
<td>Current Data</td>
<td>Spot Bonus</td>
<td>28-Mar-2014</td>
<td>1500€</td>
</tr>
<tr>
<td>Delta</td>
<td>Delete</td>
<td>Spot Bonus</td>
<td>31-Mar-2014</td>
</tr>
<tr>
<td>Delta</td>
<td>Insert</td>
<td>Spot Bonus</td>
<td>28-Mar-2014</td>
</tr>
</tbody>
</table>

Figure 66: Changing Key Field of Pay Compensation Non-Recurring

4.7.2 Period-Based Delta Transmission with the CompoundEmployee API

Period-based delta transmission of the CompoundEmployee API is for consumers that aren’t able to process effective-dated entities and future time slices.

Some typical scenarios for period-based delta transmission are:

- Monthly payroll processing, where the payroll provider is interested in data relevant for the current payroll period and retroactive changes. Whereas future dated changes are considered in a future payroll run, meaning they aren’t added to the query response for the current period.
- Daily synchronization, where the consumer is interested in data that becomes effective on this day.

Period-based delta transmission enables consumers to retrieve only changed data that is relevant for the given evaluation period. The following kinds of changes are relevant for period-based delta transmission:

- Changes on effective-dated entities with a validity in the past or with a validity overlapping with the given period and happened after the last synchronization
- Changes on effective-dated entities done in the past and where the start date falls into the given period
- Changes on non-effective-dated entities happened after the last synchronization

Changes on future dated records of effective-dated entities aren’t considered as long as their validity time frame is outside of the evaluation period.

The advantage for consumers of this approach is that only effective-dated entities are replicated that are relevant for the specified period or that include retroactive changes. Future dated changes are omitted in that feature. Future dated records of effective-dated entities will be replicated once the consumer calls the CompoundEmployee API for a period in which these changes become effective.
4.7.2.1 Enabling Period-Based Delta Transmission

Period-based delta transmission can be enabled by providing the start date and the end date of the period together with the date of the last synchronization when calling the CompoundEmployee API.

As is usual for delta transmission, the last synchronization date must be adjusted with every call. Furthermore, the periods must not overlap, so with every call another period needs to be requested. Ensure that these periods are disjunctive and starting one after the other so that there are no gaps. The length of the period isn’t limited; it’s even possible to do a daily synchronization using the same date as start and end date of the period.

In case the same period must be processed several times, the parameter isNotFirstQuery must be used.

4.7.2.2 End Date Handling

Since period-based delta transmission is designed for consumers that aren’t able to handle effective-dated entities and future changes, it ignores end dates of time slices that are outside of the given period.

No end dates for that are outside of the given period are contained in the CompoundEmployee API response. Only if a time slice of an effective-dated object ends before or within the period, the end date is also part of the response.

4.7.2.3 Parameter isNotFirstQuery

Use the isNotFirstQuery parameter to call the CompoundEmployee API multiple times for the same period.

Consumers commonly call the CompoundEmployee API once a period using period-based delta transmission to get data that becomes effective for that period. This makes sense in some cases especially when using a short period (for example, daily synchronization), but not for long periods (for example, monthly), where more regular replication of changes might be needed. For such scenarios, period-based delta transmission offers the additional parameter isNotFirstQuery, which enables the consumer to call the CompoundEmployee API multiple times for the same period. If this parameter is included in the request, the API only considers data changes that were entered into the system since the last synchronization date and that affect the provided period or the time before that period. Unchanged data, even if it becomes effective in the period, isn’t replicated anymore.

The following example shows the response of the CompoundEmployee API depending on the value of the parameter for a subsequent time slice effective in the period.
Example: Parameter `isNotFirstQuery` Isn’t Included in the Request

In this case, the parameter `isNotFirstQuery` is set to false. Since the subsequent time slice becomes effective in the period, period-based delta transmission returns the subsequent time slice with the action code `CHANGE` and indicates that field value changed from the previous value A to the new value B.
Example: Parameter `isNotFirstQuery` Is Included in the Request

In this case, the parameter `isNotFirstQuery` is set to `true`. Since the subsequent time slice wasn’t changed since the last API call, period-based delta transmission doesn’t return the subsequent time slice, because the API assumes that this time slice was already replicated in a previous run to the consumer.

4.7.2.4 Use Cases for Period-Based Delta Transmission

Here are some use cases showing the result of period-based delta transmission for different situations and changes of effective-dated entities.

The diagrams show the data as it was at the last synchronization run, the data how it is currently in the system, and the resulting delta time slices including their action codes. The use cases are based on an effective-dated entity such as job or personal information.

4.7.2.4.1 First Time Slice Starting After the Provided Period

This is the handling when a new time slice starts after the provided period.

The employee has a time slice starting after the period. There is no data before this time slice.
Since there is no relevant data before and during the provided period, period-based delta transmission does not return any data for the entity. The data is not relevant for the consumer yet and will be replicated in a later synchronization run.

### 4.7.2.4.2 First Time Slice Effective in the Provided Period

This is the handling when a time slice starts in the provided period.

The employee has a time slice starting in the period and there is no previous time slice.
Figure 68: Request for Period Delta Transmission

Period-based delta transmission returns the time slice of the entity because it becomes effective in the provided period. The action code is INSERT since it is the first time for this effective-dated entity to be replicated to the consumer.

4.7.2.4.3 Subsequent Time Slice Effective in the Provided Period

This is the handling when a value changes in the provided period.

The employee already has data for the entity and a subsequent time slice becomes effective in the provided period. The difference between the two time slices is the value of a field that changes from A to B.
Period-based delta transmission returns the subsequent time slice because it becomes effective in the provided period. The action code is CHANGE since the first time slice with value A was already replicated to the consumer in the past. The consumer is now informed about the change of the value from A to B starting with the start date of the second time slice.

Figure 69: Request for Period Delta Transmission
4.7.2.4.4 Change of Time Slice Starting Before the Provided Period

This is the handling when a time slice starting before the provided period is changed.

Data at Last Synchronization

Field = A

Current Data

Field = B

Period Delta Result

Change: Field = B (previous = A)

Figure 70: Request for Period Delta Transmission

Period-based delta transmission returns the time slice because the change affects the provided period. The action code is CHANGE since the time slice with value A was already replicated to the consumer.
4.7.2.4.5 Change of Time Slice Starting and Ending Before the Provided Period

This is the handling when a time slice starting and ending before the provided period is changed.

**Data at Last Synchronization**

- **FromDate**: Field = A
- **ToDate**: Field = B

**Current Data**

- **Field = C**
- **Field = B**

**Period Delta Result**

- **Change: Field = C**
  - (previous = A)
- **No Change: Field = B**

Figure 71: Request for Period Delta Transmission

Period-based delta transmission returns the time slice because this is a retro-active change that needs to be replicated to the consumer even if it is not relevant for the given period. The action code is CHANGE because it was already replicated to the consumer in the past with value A. The changed field is exposed with new value C and previous value A.

4.7.2.4.6 Change of Subsequent Time Slice Effective in the Provided Period

This is the handling when a subsequent time slice becomes effective and is changed in the provided period.

The employee already has data for the entity, and a subsequent time slice becomes effective in the provided period. Additionally the subsequent time slices have been changed since the last synchronization and a field was changed from value B to C.
Period-based delta transmission returns the subsequent time slice because it becomes effective in the provided period. The action code is CHANGE since the first time slice with value A was already replicated to the consumer in the past. Since the subsequent time slice was not replicated to the consumer yet, the consumer is not aware of the previous value B that was valid at the last synchronization date. Therefore the consumer is informed about the change of the field value from A to C starting at the start date of the second time slice.

4.7.2.4.7 Deletion of Subsequent Time Slice Starting After the Provided Period

This is the handling when a subsequent time starting after the provided period is deleted.

The employee has already data for the entity and there is a subsequent time slice starting after the provided period. The data has already been replicated in the past using period-based delta transmission and the subsequent time slice was ignored so far since it starts in future. Now the subsequent time slice is deleted.
In this case, period-based delta transmission does not return any data because the deleted subsequent time slice has not yet been replicated to the consumer. Since the consumer is not aware of the subsequent time slice, there is no need to get the information about the deleted time slice.

### 4.8 How the CompoundEmployee API Delta Transmission Mode Handles Foundation Objects

Here are some things you should know about how the delta transmission mode of the CompoundEmployee API handles foundation objects.

#### 4.8.1 External Codes

Look at how external codes of foundation objects are handled in the delta transmission mode of the CompoundEmployee API.

The API always returns the current value of the external code. Consequently, changes of external codes aren’t communicated by the delta mode. This also implies that the change of an external code never triggers an employee extraction.
4.8.1.1 Attributes of Effective-Dated Foundation Objects

The CompoundEmployee API exposes a few additional fields from foundation objects.

## Additional Foundation Object Fields Exposed in the API

<table>
<thead>
<tr>
<th>Foundation Object</th>
<th>Field</th>
</tr>
</thead>
<tbody>
<tr>
<td>Event Reason</td>
<td>Event</td>
</tr>
<tr>
<td></td>
<td>Employee Status</td>
</tr>
<tr>
<td></td>
<td>Payroll Event</td>
</tr>
<tr>
<td>Legal Entity</td>
<td>Country/Region</td>
</tr>
<tr>
<td>Cost Center</td>
<td>External Object ID</td>
</tr>
<tr>
<td>Frequency</td>
<td>Annualization Factor</td>
</tr>
<tr>
<td></td>
<td>Pay Frequency</td>
</tr>
</tbody>
</table>

## Critical Changes

Changes of these fields in the foundation objects are critical. If the fields are changed either using Make correction or by inserting or deleting time slices, this results in inaccurate calculation of previous values or inaccurate determination of action codes. The reason is that the API doesn't consider the history of the foundation objects. Therefore, the calculation of the snapshot data is already based on the new values of the foundation objects. That's why the delta that results from the comparison of the snapshot data and the current data isn't the real delta as it would be required for correct synchronization of Employee Central and the consumer system. In this case, manual intervention is required and the consumer system must be synchronized using full-transmission mode.

### 4.8.1.1.1 Current and Previous Values of Attributes

The calculation of the current and previous values considers the current time slices of the foundation objects.

Time slices of effective-dated foundation objects might not exactly match the employee information time slices. Here are some examples of current and previous values calculated for effective-dated foundation objects with the attributes listed in the previous section.
Example 1

<table>
<thead>
<tr>
<th>Data Change</th>
<th>PE=X</th>
<th>PE=Y</th>
</tr>
</thead>
</table>

Figure 74: FO Event Reason *Data Change* and Attribute *Payroll Event (PE)* – Job Slices Before and After Change

Job slices before the change:

A [PE=X]  B [PE=Y]

Job slices after the change:

A [PE=X]  B [PE=Y]


Figure 75: Job Delta to Be Transferred

For the 2nd time slice, payroll event Y is communicated with X as previous value.

Example 2

<table>
<thead>
<tr>
<th>Data Change</th>
<th>PE=X</th>
<th>PE=Y</th>
</tr>
</thead>
</table>

Figure 76: FO Event Reason *Data Change* and Attribute *Payroll Event (PE)*

A [PE=X]  B [PE=Y]

Figure 77: Job Slices Before Change

A [PE=X]

Figure 78: Job Slices After Change


Figure 79: Job Delta to Be Transferred

For the 2nd time slice, payroll event X is communicated with Y as previous value.
Example 3

<table>
<thead>
<tr>
<th>Delete Change</th>
<th>PE=X</th>
<th>PE=Y</th>
</tr>
</thead>
</table>

Figure 80: FO Event Reason Data Change and Attribute Payroll Event (PE) – Job Slices Before and After Change

Job slices before the change:

A [PE=X]  B [PE=Y]

Job slices after the change:

A [PE=X]  B [PE=X]


Figure 81: Job Delta to Be Transferred

For the 3rd time slice, the action code is CHANGE and payroll event X is communicated as current value and Y as previous value.

4.9 How the CompoundEmployee API Delta Transmission Mode Reacts to Data Purge

Data Retention Management allows purging of transactional data and audit data independently, with different retention periods. That’s why the CompoundEmployee API must be able to handle situations where transactional data was purged and audit data is still there.

In the delta transmission mode, the CompoundEmployee API determines the retention times of transactional data for each employee and entity. All records that are outside of the respective retention period of the underlying entity are ignored. Delta is only calculated for the records that are valid in the retention period. For the period in which data was purged and for the following day, no delta is calculated.

This means that the following changes aren’t exposed by delta calculation:

- New records that are valid outside of the retention period
- Changed or deleted records that were re-created after purge and are valid outside of the retention period

The API also checks for each entity whether the provided last_modified_on date is within the audit retention time of the entity. If this is not the case for one or more entities, CompoundEmployee API returns an error for the employee, indicating that the provided date isn’t allowed.

Example

`<log>
  <log_item>
    <person_id>4711</person_id>
    <person_id_external>Steve</person_id_external>
  </log_item>
</log>`
The audit retention time of phone information is configured to 6 months. The audit purge was executed for this employee on June 30, and the audit records of all phone information changes before December 30 were deleted. CompoundEmployee API won’t support delta queries for this employee that have a `last_modified_on` date before December 30.

In period delta mode, also the provided `fromDate` is validated against the retention periods of the requested entities. When the `fromDate` is within the purge period of an entity for an employee, CompoundEmployee API returns an error for this employee.

**Example**

```
<log>
  <log_item>
    <person_id>4711</person_id>
    <person_id_external>Steve</person_id_external>
    <code>COMPOUND_EMPLOYEE/EMPLOYEE_ERROR</code>
    <severity>ERROR</severity>
    <message_text>
      Data for user id Steve can't be returned: Please see log items for more information.
    </message_text>
  </log_item>
  <log_item>
    <person_id>4711</person_id>
    <person_id_external>Steve</person_id_external>
    <code>COMPOUND_EMPLOYEE/LAST_MODIFIED_ON_IN_AUDIT_PURGE_PERIOD</code>
    <severity>ERROR</severity>
    <message_text>
      The provided last_modified_on is outside of the audit retention period of entity "phone_information" that starts on 2016-12-30T23:00:00.000Z. Please use a last_modified_on later than 2016-12-30T23:00:00.000Z.
    </message_text>
  </log_item>
</log>
```

In period delta mode, also the provided `fromDate` is validated against the retention periods of the requested entities. When the `fromDate` is within the purge period of an entity for an employee, CompoundEmployee API returns an error for this employee.

```
<log>
  <log_item>
    <person_id>4711</person_id>
    <person_id_external>Steve</person_id_external>
    <code>COMPOUND_EMPLOYEE/EMPLOYEE_ERROR</code>
    <severity>ERROR</severity>
    <message_text>
      Data for user id Steve can't be returned: Please see log items for more information.
    </message_text>
  </log_item>
  <log_item>
    <person_id>4711</person_id>
    <person_id_external>Steve</person_id_external>
    <code>COMPOUND_EMPLOYEE/FROM_DATE_IN_PURGE_PERIOD</code>
    <severity>ERROR</severity>
    <message_text>
      The provided fromDate is outside of the retention period of entity "address_information" that starts on 2017-01-01. Please use a fromDate later than or equal to 2017-01-01.
    </message_text>
  </log_item>
</log>
```
4.10 Special Handling for the CompoundEmployee API Delta Transmission Mode

Look at some behaviors that apply to the delta transmission mode of the CompoundEmployee API.

Segments Supporting Delta Transmission [page 157]
In the delta transmission mode, the CompoundEmployee API supports many segments, but not all of them.

Segments Not Supporting Delta Transmission [page 159]
Delta transmission is not supported by these segments of the CompoundEmployee API.

Personal Documents with Duplicate Semantic Key [page 159]
How the CompoundEmployee API handles the personal_documents_information segment in the delta transmission mode.

Support of Specific Attributes [page 160]
Delta mode does not consider changes of attributes that are retrieved from table USERS_SYS_INFO.

Issues When Using Select Conditions in the Delta Transmission Mode [page 160]
You can get unexpected results if the CompoundEmployee API consumer forwards data to other systems according to specific employee attributes such as country/region, legal entity, employee class, or pay group.

4.10.1 Segments Supporting Delta Transmission

In the delta transmission mode, the CompoundEmployee API supports many segments, but not all of them.

Segments (Not) Supporting Delta Transmission

For some segments, no audit information is written or there’s no appropriate HRIS element model. In this case, no historic data can be read and consequently no delta can be calculated. Affected segments are all MDF segments (deduction_recurring, deduction_non_recurring, payment_information, alternative_cost_distribution) as well as accompanying_dependent.

<table>
<thead>
<tr>
<th>Segment</th>
<th>Supported</th>
</tr>
</thead>
<tbody>
<tr>
<td>person</td>
<td>Yes</td>
</tr>
<tr>
<td>personal_information</td>
<td>Yes</td>
</tr>
<tr>
<td>personal_information_{country_code}</td>
<td>Yes</td>
</tr>
<tr>
<td>address_information</td>
<td>Yes</td>
</tr>
<tr>
<td>email_information</td>
<td>Yes</td>
</tr>
<tr>
<td>phone_information</td>
<td>Yes</td>
</tr>
<tr>
<td>Segment</td>
<td>Supported</td>
</tr>
<tr>
<td>------------------------------------</td>
<td>-----------</td>
</tr>
<tr>
<td>person_relation</td>
<td>Yes</td>
</tr>
<tr>
<td>employment_information</td>
<td>Yes</td>
</tr>
<tr>
<td>global_assignment_information</td>
<td>Yes</td>
</tr>
<tr>
<td>job_information</td>
<td>Yes</td>
</tr>
<tr>
<td>alternative_cost_distribution</td>
<td>No</td>
</tr>
<tr>
<td>alternative_cost_distribution_item</td>
<td>No</td>
</tr>
<tr>
<td>compensation_information</td>
<td>Yes</td>
</tr>
<tr>
<td>paycompensation_recurring</td>
<td>Yes</td>
</tr>
<tr>
<td>paycompensation_non_recurring</td>
<td>Yes</td>
</tr>
<tr>
<td>payment_information</td>
<td>No</td>
</tr>
<tr>
<td>accompanying_dependent</td>
<td>No</td>
</tr>
<tr>
<td>job_relation</td>
<td>Yes</td>
</tr>
<tr>
<td>direct_deposit</td>
<td>Yes</td>
</tr>
<tr>
<td>national_id_card</td>
<td>Yes</td>
</tr>
<tr>
<td>deduction_recurring</td>
<td>No</td>
</tr>
<tr>
<td>deduction_recurring_item</td>
<td>No</td>
</tr>
<tr>
<td>deduction_non_recurring</td>
<td>No</td>
</tr>
<tr>
<td>PaymentInformationV3</td>
<td>No</td>
</tr>
<tr>
<td>ItDeclaration</td>
<td>No</td>
</tr>
</tbody>
</table>

**Parent topic:** Special Handling for the CompoundEmployee API Delta Transmission Mode [page 157]

### Related Information

- Segments Not Supporting Delta Transmission [page 159]
- Personal Documents with Duplicate Semantic Key [page 159]
- Support of Specific Attributes [page 160]
- Issues When Using Select Conditions in the Delta Transmission Mode [page 160]
4.10.2 Segments Not Supporting Delta Transmission

Delta transmission is not supported by these segments of the CompoundEmployee API.

Delta transmission doesn't support the MDF object PaymentInformation (the old payment information entity) and the HRIS object accompanying_dependent.

**i Note**

accompanying_dependent is no longer supported by the API at all.

One time deductions (or deduction_non_recurring) aren't supported in period-based delta transmission in the Q4 2015 release. Therefore, if one-time deductions are part of the SELECT statement for period-based delta transmission, processing of the query is terminated and an error message is given.

Parent topic: Special Handling for the CompoundEmployee API Delta Transmission Mode [page 157]

Related Information

Segments Supporting Delta Transmission [page 157]
Personal Documents with Duplicate Semantic Key [page 159]
Support of Specific Attributes [page 160]
Issues When Using Select Conditions in the Delta Transmission Mode [page 160]

4.10.3 Personal Documents with Duplicate Semantic Key

How the CompoundEmployee API handles the personal_documents_information segment in the delta transmission mode.

If the semantic keys of multiple documents are identical, the delta query mode for the personal_documents_information segment can't be calculated.

If the semantic keys of multiple documents are identical, the whole person isn't rendered and an appropriate error message is returned.

The semantic key of the personal_documents_information segment is made up of the following keys:

- country
- document_type
- document_number

Parent topic: Special Handling for the CompoundEmployee API Delta Transmission Mode [page 157]
4.10.4 Support of Specific Attributes

Delta mode does not consider changes of attributes that are retrieved from table USERS_SYS_INFO. The following attributes are affected:

Table 17: Attributes Retrieved from USERS_SYS_INFO Table

<table>
<thead>
<tr>
<th>Segment</th>
<th>Attribute</th>
</tr>
</thead>
<tbody>
<tr>
<td>person</td>
<td>logon_user_id</td>
</tr>
<tr>
<td></td>
<td>logon_user_name</td>
</tr>
<tr>
<td></td>
<td>logon_user_is_active</td>
</tr>
<tr>
<td>employment_information</td>
<td>direct_reports</td>
</tr>
</tbody>
</table>

For these attributes, the current value is always returned in the response message. Consequently, no delta information can be provided.

Parent topic: Special Handling for the CompoundEmployee API Delta Transmission Mode [page 157]

4.10.5 Issues When Using Select Conditions in the Delta Transmission Mode

You can get unexpected results if the CompoundEmployee API consumer forwards data to other systems according to specific employee attributes such as country/region, legal entity, employee class, or pay group.

The CompoundEmployee API delta mode is optimized for a single consumer that processes all data (for example, a single payroll system). Deltas are communicated with regard to all data records of the employee. In
In general, the CompoundEmployee API supports filters (selection parameters) on different attributes, so that employees can be extracted based on their values for these attributes. However, these filters are always applied on the employee as a whole. In this chapter, some examples are given. Although they refer to specific attributes, the same facts apply to the other mentioned attributes as well.

Example 1

Let’s say the consumer has called the delta transmission mode with the `SELECT` condition that the company territory code is `USA`. Before the transfer, CompoundEmployee API doesn’t return any data. After the transfer, the API returns the employee with all their data. However, only the second job slice has the action code `CHANGE`, the other job slices as well as all other segments have the action code `NO_CHANGE`.

In cases where US payroll is handled by a system other than German payroll, this can lead to problems since the employee doesn’t exist in the US payroll system yet. Thus, the information `NO_CHANGE` on segments such as `personal_information` and `address_information` can be misleading.

Example 2

The constellation is the same as in the first example. The CompoundEmployee API is called in delta mode with the selection condition that the company territory code is `Germany`. Even though the employee doesn’t work in Germany anymore, they’re selected for the German payroll, and deltas are communicated.

Example 3

In this example, the consumer uses different payroll systems and processes employees according to the pay group they belong to. Employees belonging to pay group 1 are handled in a different system than employees belonging to pay group 2. The Compound Employee API request is formed accordingly. For example, the
consumer processing employees of pay group 2 calls the `CompoundEmployee` API delta mode with pay group 2 as selection parameter.

The consumer responsible for pay group 2 receives the employee because one of the job slices contains pay group 2 as value. However, after the change of the pay group information to pay group 1, the consumer doesn’t receive the employee anymore. This is problematic in case the consumer completely relies on the delta transmission. The payroll system for pay group 2 isn’t informed that the employee’s pay group has changed. It won’t be informed about future changes either.

**Parent topic:** Special Handling for the `CompoundEmployee` API Delta Transmission Mode [page 157]

**Related Information**

- Segments Supporting Delta Transmission [page 157]
- Segments Not Supporting Delta Transmission [page 159]
- Personal Documents with Duplicate Semantic Key [page 159]
- Support of Specific Attributes [page 160]
5 Appendix

Some other information about the CompoundEmployee API that you might find useful.

Transient Fields [page 163]
Here’s a list of transient fields, which are configurable, but are ignored by the CompoundEmployee API.

Example: Describe and DescribeEx [page 165]
Here’s an example of how the CompoundEmployee API operations describe and describeEx return metadata such as <field> elements for segment names or custom fields.

Example: Sample Query Requests [page 167]
Here are some examples to help you make CompoundEmployee API query request calls.

Example: Sample Query Responses [page 168]
Some examples of what the CompoundEmployee API query response might look like.

5.1 Transient Fields

Here’s a list of transient fields, which are configurable, but are ignored by the CompoundEmployee API.

<table>
<thead>
<tr>
<th>Segment Name</th>
<th>Transient Field</th>
</tr>
</thead>
<tbody>
<tr>
<td>person</td>
<td>wfConfig</td>
</tr>
<tr>
<td>personal_information</td>
<td>wfConfig</td>
</tr>
<tr>
<td>person_information_country</td>
<td>wfConfig</td>
</tr>
<tr>
<td>address_information</td>
<td>wfConfig</td>
</tr>
<tr>
<td>formatted_address</td>
<td></td>
</tr>
<tr>
<td>phone_information</td>
<td>wfConfig</td>
</tr>
<tr>
<td>email_information</td>
<td>wfConfig</td>
</tr>
<tr>
<td>employment_information</td>
<td>wfConfig</td>
</tr>
<tr>
<td>person_id_external</td>
<td></td>
</tr>
<tr>
<td>attachment_id</td>
<td></td>
</tr>
<tr>
<td>company</td>
<td></td>
</tr>
<tr>
<td>isSecondaryAllSfProcesses</td>
<td></td>
</tr>
<tr>
<td>job_information</td>
<td>wfConfig</td>
</tr>
<tr>
<td>hr_manager</td>
<td></td>
</tr>
<tr>
<td>country_of_company</td>
<td></td>
</tr>
<tr>
<td>Segment Name</td>
<td>Transient Field</td>
</tr>
<tr>
<td>-----------------------------------</td>
<td>-----------------------------------</td>
</tr>
<tr>
<td>matrix_manager</td>
<td></td>
</tr>
<tr>
<td>second_manager</td>
<td></td>
</tr>
<tr>
<td>custom_manager</td>
<td></td>
</tr>
<tr>
<td>additional_manager</td>
<td></td>
</tr>
<tr>
<td>timeInPosition</td>
<td></td>
</tr>
<tr>
<td>timeInJob</td>
<td></td>
</tr>
<tr>
<td>timeInCompany</td>
<td></td>
</tr>
<tr>
<td>timeInLocation</td>
<td></td>
</tr>
<tr>
<td>timeInDepartment</td>
<td></td>
</tr>
<tr>
<td>timeInPayScaleLevel</td>
<td></td>
</tr>
<tr>
<td>compensation_information</td>
<td>wfConfig</td>
</tr>
<tr>
<td>paycompensation_recurring</td>
<td>wfConfig</td>
</tr>
<tr>
<td></td>
<td>no_changes_until_date</td>
</tr>
<tr>
<td>paycompensation_non_recurring</td>
<td>wfConfig</td>
</tr>
<tr>
<td>job_relation</td>
<td>wfConfig</td>
</tr>
<tr>
<td>global_assignment_information</td>
<td>wfConfig</td>
</tr>
<tr>
<td></td>
<td>wfConfig</td>
</tr>
<tr>
<td>direct_deposit</td>
<td>wfConfig</td>
</tr>
<tr>
<td>national_id_card</td>
<td>wfConfig</td>
</tr>
<tr>
<td>person_relation</td>
<td>wfConfig</td>
</tr>
<tr>
<td></td>
<td>related_person_id_external</td>
</tr>
</tbody>
</table>

### Related Information

Dynamic Field List [page 44]
5.2 Example: Describe and DescribeEx

Here's an example of how the CompoundEmployee API operations describe and describeEx return metadata such as <field> elements for segment names or custom fields.

Describe Query

```xml
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
    xmlns:urn="urn:sfobject.sfapi.successfactors.com">
    <soapenv:Header/>
    <soapenv:Body>
        <urn:describeSFObjects>
            <urn:type>CompoundEmployee</urn:type>
        </urn:describeSFObjects>
    </soapenv:Body>
</soapenv:Envelope>
```

DescribeEx Query

```xml
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
    xmlns:urn="urn:sfobject.sfapi.successfactors.com">
    <soapenv:Header/>
    <soapenv:Body>
        <urn:describeSFObjectsEx>
            <urn:type>CompoundEmployee</urn:type>
        </urn:describeSFObjectsEx>
    </soapenv:Body>
</soapenv:Envelope>
```

Query Response

```xml
<result>
    <type>compoundemployee</type>
    
    <field> <!-- field used in WHERE clause -->
        <name>person_id_external</name>
        <dataType>string</dataType>
        <required>false</required>
        <filterable>true</filterable>
    </field>
</result>
```
<field> <!-- segment names in the SELECT clause -->
  <name>personal_information</name>
  <dataType>string</dataType>
  <required>false</required>
  <filterable>false</filterable>
  <insertable>false</insertable>
  <selectable>true</selectable>
  <sortable>false</sortable>
  <supportInOperator>false</supportInOperator>
  <supportLikeOperator>false</supportLikeOperator>
  <updateable>false</updateable>
  <upsertable>false</upsertable>
  <constrainable>false</constrainable>
</field>

<field> <!-- standard fields contained in the result structure -->
  <name>/person/personal_information/first_name</name>
  <dataType>string</dataType>
  <label>
    <value>First Name</value>
    <locale>en_US</locale>
    <mime-type>text/plain</mime-type>
  </label>
  <maxlength>128</maxlength>
  <required>false</required>
  <filterable>false</filterable>
  <insertable>false</insertable>
  <selectable>false</selectable>
  <sortable>false</sortable>
  <supportInOperator>false</supportInOperator>
  <supportLikeOperator>false</supportLikeOperator>
  <updateable>false</updateable>
  <upsertable>false</upsertable>
  <constrainable>false</constrainable>
</field>

<field> <!-- custom fields contained in the result structure -->
  <name>/person/personal_information/custom_string1</name>
  <dataType>string</dataType>
  <label>
    <value>Nickname</value>
    <locale>en_US</locale>
    <mime-type>text/plain</mime-type>
  </label>
  <maxlength>128</maxlength>
  <required>false</required>
  <filterable>false</filterable>
  <insertable>false</insertable>
  <selectable>false</selectable>
  <sortable>false</sortable>
  <supportInOperator>false</supportInOperator>
  <supportLikeOperator>false</supportLikeOperator>
  <updateable>false</updateable>
  <upsertable>false</upsertable>
  <constrainable>false</constrainable>
</field>
5.3 Example: Sample Query Requests

Here are some examples to help you make CompoundEmployee API query request calls.

Example 1

Select an employee by the external person ID.

```xml
<urn:query>
  <urn:queryString>
    select person,
    personal_information,
    address_information,
    email_information,
    phone_information,
    employment_information,
    job_information,
    compensation_information,
    paycompensation_recurring,
    paycompensation_non_recurring,
    direct_deposit,
    national_id_card,
    payment_information
    from CompoundEmployee
    where person_id_external = 'cgrant'
  </urn:queryString>
  <urn:param>
    <urn:name>maxRows</urn:name>
    <urn:value>50</urn:value>
  </urn:param>
</urn:query>
```

Example 2

Select all employees that have been changed between September 10 and September 18, 2019 and return the entities person and personal_information only.

```xml
<urn:query>
  <urn:queryString>
    select person, personal_information
    from CompoundEmployee
    where last_modified_on > to_date('10/09/19', 'DD/MM/YY')
    and last_modified_on < to_date('18/09/19', 'DD/MM/YY')
  </urn:queryString>
</urn:query>
```
Example 3

Select all employees to which the following conditions apply:

- They have a job in company CompanyABC.
- They have a job at a company located in country/region USA at any point in time.
- They have a job of employee class E.

Return all effective-dated records with end date that’s on or after September 10, 2019. Return a maximum of 10 employees.

Sample Code

```xml
<urn:queryString>
  select person,
  personal_information,
  address_information,
  email_information,
  phone_information,
  employment_information,
  job_information,
  compensation_information,
  paycompensation_recurring,
  paycompensation_non_recurring,
  direct_deposit,
  national_id_card,
  payment_information
  from CompoundEmployee
  where company='Company ABC'
  and company_territory_code='USA'
  and employee_class='E'
  and effective_end_date>=to_date('2019-09-10','YYYY-MM-DD')
</urn:queryString>
<urn:param>
  <urn:name>maxRows</urn:name>
  <urn:value>10</urn:value>
</urn:param>
</urn:query>
```

5.4 Example: Sample Query Responses

Some examples of what the CompoundEmployee API query response might look like.

```xml
<queryResponse xmlns="urn:sobject.sfapi.successfactors.com"
  xmlns:ns2="urn:fault.sfapi.successfactors.com">
  <result>
    <sobject>
      <id>91</id>
      <type>CompoundEmployee</type>
      <person>
```
<end_date>2010-12-31</end_date>

<last_modified_on>2013-02-11T11:45:11.000Z</last_modified_on>
</address_information>

<address_information>
  <address_type>home</address_type>
  <address1>2134, Forest Hill Rd</address1>
  <city>New York</city>
  <county>Richmond County</county>
  <zip_code>10301</zip_code>
  <country>USA</country>
  <start_date>2012-11-24</start_date>
  <end_date>2012-12-31</end_date>
  <last_modified_on>2013-02-11T11:45:11.000Z</last_modified_on>
</address_information>

<address_information>
  <address_type>home</address_type>
  <address1>Glockengasse 4711</address1>
  <city>Köln</city>
  <county>Köln</county>
  <zip_code>50667</zip_code>
  <country>DEU</country>
  <start_date>2012-11-23</start_date>
  <end_date>2012-11-23</end_date>
  <last_modified_on>2013-02-11T11:45:11.000Z</last_modified_on>
</address_information>

<address_information>
  <address_type>home</address_type>
  <address1>1128a, 5th Avenue</address1>
  <city>New York</city>
  <county>New York County</county>
  <zip_code>10301</zip_code>
  <country>USA</country>
  <start_date>2011-01-01</start_date>
  <end_date>2011-03-30</end_date>
  <last_modified_on>2013-02-11T11:45:11.000Z</last_modified_on>
</address_information>

<address_information>
  <address_type>business</address_type>
  <address1>1500 Fashion Island Blvd.</address1>
  <address2>Ste. 300</address2>
  <city>San Mateo</city>
  <zip_code>94404</zip_code>
  <country>USA</country>
  <start_date>2011-01-01</start_date>
  <end_date>2011-12-31</end_date>
  <last_modified_on>2013-02-01T15:02:37.000Z</last_modified_on>
</address_information>

<address_information>
  <address_type>vacation</address_type>
  <address1>452 Holiday Road</address1>
  <city>Scottsdale</city>
  <county>Maricopa</county>
  <zip_code>85252</zip_code>
  <country>USA</country>
  <start_date>2011-01-01</start_date>
  <end_date>2011-03-30</end_date>
  <last_modified_on>2013-02-01T15:02:37.000Z</last_modified_on>
</address_information>

<custom_string1>480-555-2144</custom_string1>
</address_information>

<address_information>
  <address_type>business</address_type>
  <address1>1500 Fashion Island Blvd.</address1>
</address_information>
Employee Central Compound Employee API

Appendix

PUBLIC

173
<start_date>2010-08-16</start_date>
<end_date>2010-12-31</end_date>
<last_modified_on>2011-01-14T00:46:58.000Z</last_modified_on>
<custom_string1>480-555-2144</custom_string1>
</address_information>

<phone_information>
<area_code>919</area_code>
<phone_number>565-3345</phone_number>
<phone_type>C</phone_type>
<isPrimary>false</isPrimary>
<last_modified_on>2011-05-11T04:29:29.000Z</last_modified_on>
</phone_information>

<phone_information>
<country_code>1</country_code>
<area_code>650</area_code>
<phone_number>+1-888-888-8888</phone_number>
<phone_type>B</phone_type>
<isPrimary>true</isPrimary>
<last_modified_on>2011-07-28T04:26:05.000Z</last_modified_on>
</phone_information>

<email_information>
<email_address>RevolutionUI@successfactors.com</email_address>
<email_type>B</email_type>
<isPrimary>true</isPrimary>
<last_modified_on>2012-01-09T23:09:42.000Z</last_modified_on>
</email_information>

<employment_information>
<user_id>cgrant1</user_id>
<jobNumber>1</jobNumber>
<start_date>2001-01-01</start_date>
<originalStartDate>2013-01-01</originalStartDate>
<last_modified_on>2013-02-01T13:39:08.000Z</last_modified_on>
<employment_id>409</employment_id>
</job_information>

<company>ACE_USA</company>
<business_unit>ACE_SVCS</business_unit>
<cost_center>41100</cost_center>
<is_fulltime_employee>true</is_fulltime_employee>
<pay_grade>GR-18</pay_grade>
<event_reason>PAYSCP</event_reason>
<timezone>CST</timezone>
<manager_id>dcortez1</manager_id>
<manager_person_id>143</manager_person_id>
<company_territory_code>USA</company_territory_code>
<emplStatus>A</emplStatus>
<event>12</event>
</job_information>

<job_information>
<job_title>Engineer2</job_title>
<job_code>GM</job_code>
<department>CORP</department>
<division>ENT</division>
<location>US_ATL</location>
<start_date>2012-10-04</start_date>
<end_date>9999-12-31</end_date>
<last_modified_on>2012-12-20T14:14:52.000Z</last_modified_on>
</job_information>
<start_date>2011-01-01</start_date><end_date>2012-10-03</end_date><last_modified_on>2012-12-20T14:10:09.000Z</last_modified_on><company>ACE_USA</company><business_unit>ACE_SVCS</business_unit><cost_center>41100</cost_center><is_fulltime_employee>true</is_fulltime_employee><pay_grade>GR-18</pay_grade><event_reason>PAYSCH</event_reason><timezone>CST</timezone><manager_id>dcortez1</manager_id><manager_person_id>143</manager_person_id><company_territory_code>USA</company_territory_code><emplStatus>A</emplStatus><event>12</event></job_information><job_information><job_title>General Manager, IND</job_title><job_code>GM</job_code><department>IND</department><division>IND</division><location>US_SFO</location><start_date>2005-11-01</start_date><end_date>2010-12-31</end_date><last_modified_on>2011-10-12T00:44:57.000Z</last_modified_on><company>ACE_USA</company><business_unit>ACE_IND</business_unit><cost_center>30001</cost_center><is_fulltime_employee>true</is_fulltime_employee><pay_grade>GR-18</pay_grade><event_reason>HIRENEW</event_reason><timezone>CST</timezone><manager_id>dcortez1</manager_id><manager_person_id>143</manager_person_id><company_territory_code>USA</company_territory_code><emplStatus>A</emplStatus><event>H</event></job_information><compensation_information><pay_type>S</pay_type><start_date>2011-08-01</start_date><end_date>2011-08-14</end_date><last_modified_on>2011-08-16T01:37:24.000Z</last_modified_on><pay_group>NA_GROUP</pay_group></compensation_information><compensation_information><pay_type>S</pay_type><start_date>2009-04-01</start_date><end_date>2009-03-31</end_date><last_modified_on>2011-07-27T04:46:14.000Z</last_modified_on><pay_group>NA_GROUP</pay_group></compensation_information><compensation_information><pay_type>S</pay_type><start_date>2012-10-04</start_date><end_date>9999-12-31</end_date><last_modified_on>2012-10-04T11:35:18.000Z</last_modified_on><pay_group>NA_GROUP</pay_group></compensation_information><compensation_information><pay_type>S</pay_type><start_date>2008-04-01</start_date><end_date>2009-03-31</end_date><last_modified_on>2012-10-04T11:35:18.000Z</last_modified_on><pay_group>NA_GROUP</pay_group></compensation_information>
last_modified_on>2011-07-27T04:45:09.000Z</last_modified_on>

<compensation_information>
<pay_group>NA_GROUP</pay_group>
<pay_type>S</pay_type>
<start_date>2011-07-29</start_date>
<end_date>2011-07-31</end_date>
</compensation_information>

last_modified_on>2011-08-01T21:09:40.000Z</last_modified_on>

<compensation_information>
<pay_group>NA_GROUP</pay_group>
<pay_type>S</pay_type>
<start_date>2010-04-01</start_date>
<end_date>2011-07-28</end_date>
</compensation_information>

last_modified_on>2011-07-29T23:08:39.000Z</last_modified_on>

<compensation_information>
<pay_group>NA_GROUP</pay_group>
<pay_type>S</pay_type>
<start_date>2005-11-01</start_date>
<end_date>2006-03-31</end_date>
</compensation_information>

last_modified_on>2011-07-27T04:44:01.000Z</last_modified_on>

<compensation_information>
<pay_group>NA_GROUP</pay_group>
<pay_type>S</pay_type>
<start_date>2011-08-15</start_date>
<end_date>2012-10-03</end_date>
</compensation_information>

last_modified_on>2011-08-16T01:37:24.000Z</last_modified_on>

<compensation_information>
<pay_group>NA_GROUP</pay_group>
<pay_type>S</pay_type>
<start_date>2007-04-01</start_date>
<end_date>2008-03-31</end_date>
</compensation_information>

last_modified_on>2011-07-27T04:45:36.000Z</last_modified_on>

<compensation_information>
<pay_group>NA_GROUP</pay_group>
<pay_type>S</pay_type>
<start_date>2006-04-01</start_date>
<end_date>2007-03-31</end_date>
</compensation_information>

last_modified_on>2011-07-27T04:44:32.000Z</last_modified_on>

<compensation_information>
<pay_group>NA_GROUP</pay_group>
<employment_information>

</person>
</sfobject>
<numResults>1</numResults>
<hasMore>false</hasMore>
<querySessionId>ec8084ad-8198-4963-9046-3cc8f9ec7276</querySessionId>
</result>
</queryResponse>
Important Disclaimers and Legal Information

Hyperlinks

Some links are classified by an icon and/or a mouseover text. These links provide additional information.

About the icons:

- Links with the icon 🌐: You are entering a Web site that is not hosted by SAP. By using such links, you agree (unless expressly stated otherwise in your agreements with SAP) to this:
  - The content of the linked-to site is not SAP documentation. You may not infer any product claims against SAP based on this information.
  - SAP does not agree or disagree with the content on the linked-to site, nor does SAP warrant the availability and correctness. SAP shall not be liable for any damages caused by the use of such content unless damages have been caused by SAP’s gross negligence or willful misconduct.
- Links with the icon 🏠: You are leaving the documentation for that particular SAP product or service and are entering a SAP-hosted Web site. By using such links, you agree that (unless expressly stated otherwise in your agreements with SAP) you may not infer any product claims against SAP based on this information.

Videos Hosted on External Platforms

Some videos may point to third-party video hosting platforms. SAP cannot guarantee the future availability of videos stored on these platforms. Furthermore, any advertisements or other content hosted on these platforms (for example, suggested videos or by navigating to other videos hosted on the same site), are not within the control or responsibility of SAP.

Beta and Other Experimental Features

Experimental features are not part of the officially delivered scope that SAP guarantees for future releases. This means that experimental features may be changed by SAP at any time for any reason without notice. Experimental features are not for productive use. You may not demonstrate, test, examine, evaluate or otherwise use the experimental features in a live operating environment or with data that has not been sufficiently backed up.

The purpose of experimental features is to get feedback early on, allowing customers and partners to influence the future product accordingly. By providing your feedback (e.g. in the SAP Community), you accept that intellectual property rights of the contributions or derivative works shall remain the exclusive property of SAP.

Example Code

Any software coding and/or code snippets are examples. They are not for productive use. The example code is only intended to better explain and visualize the syntax and phrasing rules. SAP does not warrant the correctness and completeness of the example code. SAP shall not be liable for errors or damages caused by the use of example code unless damages have been caused by SAP’s gross negligence or willful misconduct.

Bias-Free Language

SAP supports a culture of diversity and inclusion. Whenever possible, we use unbiased language in our documentation to refer to people of all cultures, ethnicities, genders, and abilities.