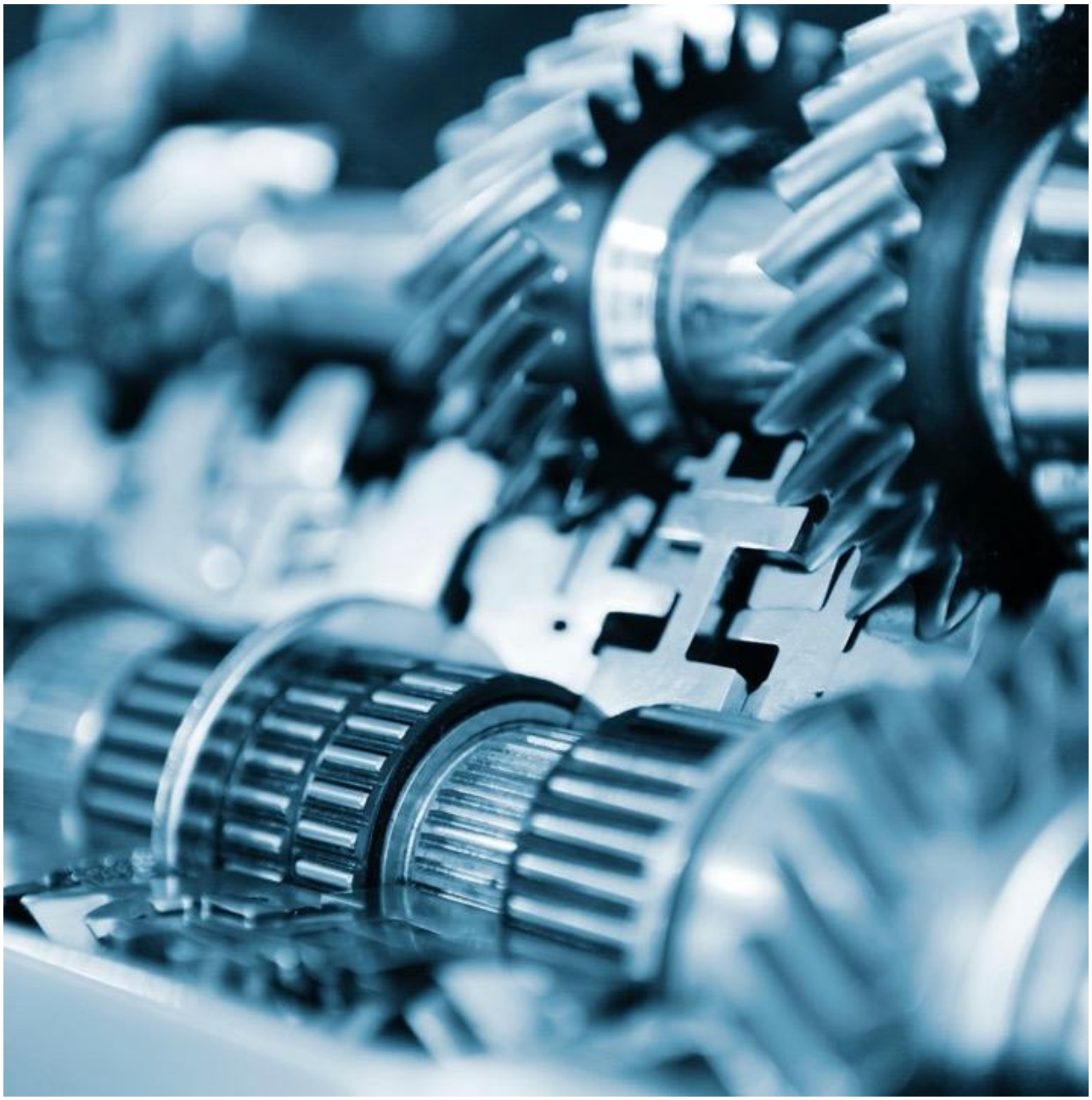
WinTid Integration Implementation documentation



Confidential

Table of Contents

1 Introduction

1.1 Goal

This document should provide necessary information to implement integrations that access WinTid via Wintid API, Wintid Integrationservice as well as file-based imports and exports using Wintid Integrationserver.

1.2 Intended audience

Developers and DevOps who need to connect programmatically to WinTid in order to integrate with other systems such as HR, ERP, Project management.

2 WinTid API

2.1 Responsible Use of API Endpoints

To ensure stable operations and optimal performance for all our customers, we require responsible and reasonable use of our API endpoints, both for data retrieval and submission. Requests should be limited to what is necessary for your use case, and unnecessary frequent requests or bulk operations should be avoided.

If usage patterns result in significantly increased resource consumption on our systems and necessitate capacity expansions, we reserve the right to evaluate how this should be addressed, including potential adjustments to the cost structure. This will be managed in dialogue with the relevant parties.

2.2 Connecting to Wintid Api

WinTid API is a .net web API that can be run as a standalone .exe which self-hosts with Kestrel, or it can be hosted inside Microsoft Internet Information Services (IIS).

2.3 Self-hosting in kestrel

By starting the wintidapi.exe application, WinTidApi starts up and will by default listen for http requests on localhost port 5000 and https on localhost port 5001.

This can be changed in appsettings.json by adding a kestrel section – for example the below section will make it listen on all interfaces ports 80/443:

```
  "Kestrel": {
      "Endpoints": {
          "Http": {
              "Url": "http://*:80"
          },
          "Https": {
              "Url": "https://*:443"
          }
      }
  }
```

Configuring https certificates and other configuration options for Kestrel are outside the scope of this document, but see e.g. https://learn.microsoft.com/en-us/aspnet/core/fundamentals/servers/kestrel/options?view=aspnetcore-8.0 for more information.

Verification that WinTid Api is running can also be done on the command line, by using curl (https://curl.se/):

```
curl http://localhost:5000/pfield/1
```


```
[
        {"Level":1,"Id":"14.2.0","Name":"14.2.0","Active":true,"ActiveFromDate":null,"ActiveToDate":null},
        {"Level":1,"Id":"14.3.0","Name":"14.3.0","Active":true,"ActiveFromDate":null,"ActiveToDate":null},
        {"Level":1,"Id":"S_1001","Name":"Product Development","Active":true,"ActiveFromDate":null,"ActiveToDate":null},
        {"Level":1,"Id":"S_1000","Name":"Project allocation time","Active":true,"ActiveFromDate":null,"ActiveToDate":null},
        {"Level":1,"Id":"100002","Name":"Internal","Active":false,"ActiveFromDate":null,"ActiveToDate":null}
]
```

2.3.1 Authentication

WinTid API supports three modes of operation with regards to authentication:

1. No authentication – all calls honored.
2. HTTP Basic authentication – with a single set of credentials that gives full access to the API
3. Api Key authentication where api keys can be created as needed, with or without expiration dates, and with access to a given role that in run gives access to a subset of api endpoints.

When WinTid is part of CGI SaaS solution the Api Key authentication must be used

Authentication is determined by the *Access* section of appsettings.json:

```
"Access": {
    "AuthRequired": false,
    "UseApiKeys": true,
    "Username": "testuser",
    "Password": "testpass"
},
```

2.3.1.1 No authentication

By leaving *AuthRequired* to *false*, WinTidApi will be “open” and not require any authentication

2.3.1.2 HTTP Basic authentication

By setting *AuthRequired* to true, leave *UseApiKeys* as false and configure a username and password, WinTidApi will require an *Authorization: Basic base64encoded(username:password)* http header in the request, or the request will be rejected.

```
   "Access": {
       "AuthRequired": true,
       "UseApiKeys": false,
       "Username": "testuser",
       "Password": "testpass"
   },
```

The above settings would require the following http header in the request:

```
Authorization: Basic dGVzdHVzZXI6dGVzdHBhc3M=
```

2.3.1.3 Api key authentication

By setting both *AuthRequired* and *UseApiKeys* to *true*, WinTidApi will require an Authorization: Bearer base64encoded(token) with a valid bearer token that gives access to a role in WinTidApi:

```
"Access": {
  "AuthRequired": true,
  "UseApiKeys" : true
},
```

Existing roles can be listed with wintid-cli:

```
wintid-cli listapirole
```

```
1 – ABSENCE
2 – DYNAMICQUERY
3 – EMPLOYEE
4 – EXPORT
5 – IMPORT
6 – JOB
7 - PROJECT
```

To see the actual endpoints in a role, add the role id with -r:

```
wintid-cli listapirole -r 5
```

```
5 – IMPORT
/import/company
/import/employeeandposition
/import/employees
/import/employeesnapshot
/import/organizationunit
```

A token can be created using wintid-cli:

```
wintid-cli addapikey -n hr_employeeimport -e 2025-12-31 -r 5
```

Where -n is the name of the key, -e is expiry date (optional) and -r is the id of the role, in this case the default import role. The above command creates an apikey called hr_employeeImport assigned to role 5 (import) with access to the import endpoints only. The output of the command contains the base-64 encoded token. **Important**: this is the only time the token is available. It is not stored in wintid, if you lose it you will need to generate a new key.

```
Key added.
Apikey 19: hr_employeeimport, Expiration: 31/12/2025, Assigned to Role id 5
Please securely store this token (it can not be retrieved again): Wg2AfPAuSzqRU8b+bpdOqg==
```

Wintid comes with a default set of roles that are stored in the dbo.wintidapi_role table.

A role can have access to endpoints, defined in dbo.wintidapi_role_endpoint table.

It is recommended to leave the existing roles unchanged and instead create new roles for your custom needs.

Currently, changes to wintidapi_rol and wintidapi_role_endpoint requires using T-SQL commands.

2.4 API Reference

WinTidApi provides simple swagger documentation for most interfaces, it can be found at /swagger/index.html

Note: Employee import, Employee snapshot import, Employee snapshot by firm import, Job import and Organization unit import do not provide correct swagger documentation due to the flexible format of those imports.

2.4.1 Errors

If a request is malformed or illegal, WintidApi will usually provide an HTTP 400 Bad Request error, with error information in the body:

```
{"Message": "PField1Id must be specified","Type": "job_error","Status": "failed"}
```

2.4.2 PField module

Allows retrieving, updating and adding PFields, stored in the pfelt1, pfelt2, pfelt3, pfelt4 and pfelt5 tables.

`HTTP GET /pfield/{pfield level}`

- returns a list of all pfields on the specified level

`HTTP GET /pfield/{pfield level}/{pfield id}`

- returns information on the pfield entry with the specified level and pfield id.

`HTTP GET /pfield/levels`

- returns a list of all pfield levels used in the Wintid instance.

`HTTP PUT /pfield/{pfield level}/{pfield id}`

- pfield data in body (see examples below)
- adds or updates a pfield with the specified level and id.

`HTTP POST /pfield/`

- pfield data in body
- adds or updates a pfield with the level and id in the body

2.4.2.1 Examples

Examples assume running locally on the server that is running WintidApi, and that it is listening on the default port 5000.

Retrieve all configured pfield levels

```
curl http://localhost:5000/pfield/levels
```

```
1,2,3,4
```

List all pfields on level 1

```
curl http://localhost:5000/pfield/1
```

```
[
    {
        "Level": 1,
        "Id": "S_1000",
```

```
        "Name": "Project",
        "Active": true,
        "ActiveFromDate": null,
        "ActiveToDate": null
    },
    {
        "Level": 1,
        "Id": "S_1001",
        "Name": "Product development",
        "Active": true,
        "ActiveFromDate": null,
        "ActiveToDate": null
    },
    {
        "Level": 1,
        "Id": "100002",
        "Name": "Internal time",
        "Active": false,
        "ActiveFromDate": null,
        "ActiveToDate": null
    },
    {
        "Level": 1,
        "Id": "1005",
        "Name": "Testing",
        "Active": false,
        "ActiveFromDate": null,
        "ActiveToDate": null
    }
]
```


Show one specific pfield

```
curl http://localhost:5000/pfield/1/S_1001
```


```
{
    "Level": 1,
    "Id": "S_1001",
    "Name": "Product Development",
    "Active": true,
    "ActiveFromDate": null,
    "ActiveToDate": null
}
```


Same example as above, returning xml instead of json when given .xml extension. In an API request, specifying the Content-type: application/json or application/xml in the request header determines the returned format.

```
curl http://localhost:5000/pfield/1/S_1001.xml
```


```
<?xml version="1.0"?>
<PField xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <Level>1</Level>
  <Id>S_1001</Id>
  <Name>Product Development</Name>
  <Active>true</Active>
  <ActiveFromDate xsi:nil="true" />
  <ActiveToDate xsi:nil="true" />
</PField>
```

Update a single pfield using json

```
curl -v -X PUT -H "Content-Type: application/json" -d '{"Level": 1,"Id": "S_1001","Name": "json PUT update","Active": true,"ActiveFromDate": null,"ActiveToDate": null}' http://localhost:5000/pfield/1/S_1001
```

Update a single pfield using json

```
curl -v -X POST -H "Content-Type: application/json" -d '{"Level":1,"Id":"S_1001","Name":"json POST update","Active":true,"ActiveFromDate":null,"ActiveToDate":null}' http://localhost:5000/pfield
```

Update a single pfield using XML

```
curl -v -X PUT -H "Content-Type: application/xml" -d '<?xml version="1.0"?><PField xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xsd="http://www.w3.org/2001/XMLSchema"><Level>1</Level><Id>S_1001</Id><Name>XML PUT update</Name><Active>true</Active><ActiveFromDate xsi:nil="true" /><ActiveToDate xsi:nil="true" /></PField>' http://localhost:5000/pfield/1/S_1001
```

Update a single pfield using XML

```
curl -v -X POST -H "Content-Type: application/xml" -d '<?xml version="1.0"?><PField xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xsd="http://www.w3.org/2001/XMLSchema"><Level>1</Level><Id>S_1001</Id><Name>XML POST update</Name><Active>true</Active><ActiveFromDate xsi:nil="true" /><ActiveToDate xsi:nil="true" /></PField>' http://localhost:5000/pfield
```

2.4.3 Job module

2.4.3.1 HTTP POST /import/job

Allows importing new and updated jobs. POST body should contain required fields similar to file-based imports (see 7.3 for a description of the possible fields). Supported formats are XML and JSON.

Example curl command to post import file in XML format:

```
curl -v -X POST -H "Content-Type: application/json" -d @importfile.xml
http://localhost:5000/import/job
```

JSON example

```
{
    "JobImport": {
        "Header": {
            "Date": "2025-02-13",
            "Sender": "Agresso",
            "Type": "Employee Import",
            "DepartmentType": "N"
        },
        "Jobs": [
            {
            "JobName": "H - Ledelse",
            "Pfield1No":"111",
            "Pfield2No":"222",
            "Pfield4No":"111",
            "JobBudget":"11,5",
            "JobPlannedStartDate":"2009-07-30",
            "JobPlannedEndDate":"2009-08-01",
            "ResponsibleEmployeeId": "26529"
            }
        ]
    }
  }
```

Notes:

- If a Pfield number is accompanied by a name, that pfield will be created if it does not exist. If it does exist it will be updated if the name is different from the existing name
- Multiple jobs can be submitted in the same call
- If a new job is not provided a job number, one will be auto-generated for it, based on the Unix Epoch of the current second combined with a counter that allows up to 100 million unique job numbers per second.

2.4.3.2 HTTP PUT / POST /job

The job endpoint allows a simple interface for adding jobs/projects to the jobb table in Wintid. Can update names on existing jobs, identified by pfield combination.

2.4.3.2.1 Example

```
curl -v -X POST -H "Content-Type: application/json" -d '{"Id": 202010061607050176,"Name": "updated job name","Pfield1Id": "12","Pfield2Id": "11","Pfield3Id": "3001"}' http://localhost:5000/job
```

2.4.4 Employee module

2.4.4.1 HTTP POST /import/employees

Allows importing new and updated employees, typically from HR systems. POST body should contain required fields similar to file-based imports, but supports JSON in addition to XML. Note that this endpoint will always create a position when importing a new employee, and will always update the current position at the specified date in the header section of the body – or todays date if not specified. If wintid is configured to create new positions upon changes to working percentage or department, changing these fields in the import will create a new position from the above date.

Example curl command to post import file in XML format

```
curl -v -X POST -H "Content-Type: application/xml" -d @importfile.xml http://localhost:5000/import/employees
```

Minimal XML example file

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<EmployeeImport>
    <Header>
        <Date>2012-10-16</Date>
        <Sender>Excel</Sender>
        <Type>Employee Import</Type>
        <DepartmentType>N</DepartmentType>
    </Header>
    <Employees>
        <Employee>
            <EmployeeId>3334</EmployeeId>
            <FirstName>Ola</FirstName>
            <LastName>Nordmann</LastName>
        </Employee>
    </Employees>
</EmployeeImport>
```

Example curl command to post import file in JSON format

```
curl -v -X POST -H "Content-Type: application/json" -d @importfile.json
http://localhost:5000/import/employees
```

Minimal json example importing 2 employees

```
{
    "EmployeeImport": {
        "Header": {
            "Date": "2012-10-16",
            "Sender": "HR",
            "Type": "Employee Import",
            "DepartmentType": "N"
        },
        "Employees": [
            {
                "EmployeeId": "3335",
                "LastName": "Nordmann",
                "FirstName": "Ola",
            },
            {
                "EmployeeId": "3336",
                "LastName": "Another",
                "FirstName": "Employee"
            }
        ]
    }
}
```

Example return values on completion of import

```
{"status":6,"transactions":1}
```

See 7.1 for a full list of fields for the Header, Employee and Position entries, as well as a description of the status codes returned.

2.4.4.2 HTTP POST import/employeesnapshot

- This endpoint allows import of new employees/positions as well as updates to exiting positions in a “snapshot” manner. This mean that the positions on an employee in WinTid will mirror the positions included in the import. A new position in the import file will be added to WinTid, and a position that is missing from the import file but exists in WinTid will be marked as deleted in WinTid.
- This import type is meant for customers who want to maintain a one-to-one mapping of WinTid positions with positions in their HR or similar systems.
- One call to the endpoint can include many employees, and each employee can include many positions.
- EmployeeId (maps to employee.external_employee_id in WinTid) is used to uniquely identify which employee should be updated. PositionId (maps to employee_position.external_position_id) is used with EmployeeId to

uniquely identify the position to update. This means that if a later call to this endpoint has changed the PositionId of a position, it will be interpreted as if the original position is deleted, and that a new position with the new PositionId is created.

- Positions in the same chain (same CorrelationId on the same employee) can not overlap in time – any change that would cause overlapping positions in the same position chain will be rejected with an error.
- If a position has its dates changed so that some existing time data ends up outside the position, that data will be visible in the Ownerless Data page in MinWinTid where it can be transferred to other positions on the employee that currently do not hold time data.
- If a position has such ownerless data outside the start- and enddates of the position, and a later update adds or changes another position so it now spans this date range AND both the original and the new/updated position have the same CorrelationId, then the time data will be automatically moved to the new position, as long as it does not have time data in the range.
- New positions added that have an existing earlier position bordering it in the same chain (prior position ends on the day before the new position starts), will have settings copied from the prior position. This includes all settings except start date and end date
- New positions added that do NOT have an existing earlier position bordering it in the same chain, will be created with default values if configured. See 7.2 for details.
- Calendar id, wage group id and fixed overtime id are all settings that include a date, and the setting applies from that date and until it is replaced by a different setting on a later date. If one of these fields is imported WITHOUT a date specified, it applies from the position start date. If an import file attempts to update one of these settings on a date that already has a different id, it will be rejected with an error – but note that calendar has some configurable functionality, see next bullet point.
- When importing an updated calendar id on a date where a position already has a calendar id, the default action is for the import of that position to fail. This behaviour can be customized by adding an entry in wt_system table, *emp_import_calendar_setting* which supports the following modes:
    - Default: Same as no setting, same functionality as specified above.
    - Ignore: Any difference in the calendar id in the import data on the same date as an existing calendar id, is ignored – WinTid value will be unchanged, but no error.
    - Overwrite: The imported value will overwrite any existing value on the same date.
- When importing an updated wage group id on a date where a position already has a wage group id, the default action is for the import of that position to fail. This behaviour can be customized by adding an entry in wt_system table, *emp_import_wagegroup_setting* which supports the following modes:
    - Default: Same as no setting, same functionality as specified above.
    - Ignore: Any difference in the wage group id in the import data on the same date as an existing wage group id, is ignored – WinTid value will be unchanged, but no error.
    - Overwrite: The imported value will overwrite any existing value on the same date.
- If an existing position has its start date moved back in time, and it had any of calendar, fixedovertime code or wage group id on the original start date – but does NOT have one on the updated startdate, then those settings are moved back in time so they are available from the updated and earlier position start date.
- Snapshot import supports setting a cutoff date. When a cutoffdate is set, then any position with an end date prior to this date will not be affected by imported positions/updates nor will they be deleted if they are not included in the import. This means that such positions, for all practical purposes, cannot be edited via snapshot import. Additionally, any new or updated position that has/is updated to its startdate before the

cutoffdate and its end date on or after the cutoffdate, will have its start date set to the cutoffdate. Enable this setting by setting the wt_system setting *snapshot_cutoff_date* to a valid yyy-MM-dd date.

NOTE: If some employees fail to import, HTTP 400 Bad Request will be returned, although other employees might have been successfully imported. The response body will contain details on errors.

The post body must contain valid XML or JSON data similar to the other file-based imports. This includes a Header followed by an Employees list of employees, which in turn can contain a list of positions. Some examples included below.

Example json import file importing one employee with 3 positions


```
{
    "EmployeeImport": {
        "Header": {
            "Date": "2021-05-26",
            "Sender": "Agresso",
            "Type": "Employee Import",
        },
        "Employees": [
            {
                "EmployeeId": "3338",
                "LastName": "Normann",
                "FirstName": "Ola",
                "EngageFromDate": "2020-01-01",
                "Email": "ola.nordmann@test.com",
                "optionalTexts":[
                        {"textID":"pfelt1","textValue":"Unit"},
                        {"textID":"pfelt2","textValue":"Test"}
                ],
                "Positions": [
                    {
                        "PositionId": "3338-1",
                        "CorrelationId": "main",
                        "DepartmentId": "82",
                        "PositionStartDate": "2020-01-01",
                        "PositionEndDate": "2020-06-30",
                        "CalendarFromDate": "2000-01-01",
                        "WageId": "UnusedWageId",
                        "WorkPercent": "100",
                        "PositionDescription": "First position in chain.",
                        "WageGroupId": "10",
                    },
                    {
                        "PositionId": "3338-2",
                        "CorrelationId": "main",
                        "DepartmentId": "82",
                        "IsPrimaryPosition": "True",
```

```
                    "PositionStartDate": "2020-07-01",
                    "CalendarFromDate": "2020-07-01",
                    "WageId": "UnusedWageId",
                    "WorkPercent": "100",
                    "PositionDescription": "second in chain, primary",
                    "WageGroupId": "10"
                },
                {
                    "PositionId": "1177",
                    "CorrelationId": "other",
                    "DepartmentId": "90",
                    "IsPrimaryPosition": "False",
                    "PositionStartDate": "2020-05-01",
                    "CalendarFromDate": "2020-05-01",
                    "WageId": "UnusedWageId",
                    "WorkPercent": "100",
                    "PositionDescription": "Some other position",
                    "WageGroupId": "10"
                }
            ]
        }
    ]
  }
}
```


Example return values on completion of import


```
{"status":6,"transactions":1}
```


See 7.1 for a full list of fields for the Header, Employee and Position entries, as well as a description of the status codes returned.

2.4.4.3 HTTP POST import/employeesnapshotbyfirm

This endpoint performs a snapshot import like the employeesnapshot import with the following differences (the data format is exactly the same as for snapshot imports):

a) The import will ignore positions in any firm *not* included in the import for snapshot functionality. This allows for performing snapshot imports on active firms without providing older positions in non-relevant firms, without them being automatically deleted.

b) Note that validations will still apply – you can not have overlapping positions in the same position chain, even if they are in different firms.

c) CutOffdate functionality is not currently supported for snapshotbyfim imports.

2.4.4.4 HTTP POST /import/employeeandposition

a) This endpoint allows import of new employees/positions as well as updates to exiting positions, including deleting existing positions. A new position in the import file (identified by the combination of employee id and position id) will be added to WinTid.

b) The endpoint supports the same data format as the /import/employeesnapshot import, but without the “snapshot” functionality. Only positions to add or update need to be included in the body of the call. Positions not included, will not be changed (with one exception explained in f)).

c) One call to the endpoint can include many employees, and each employee can include many positions.

d) Only fields included in the post data will be updated. It is not required to provide more fields than what is necessary to identify the employee and any positions (EmployeeId and PositionId), as well as the fields to update. Note that missing required fields for new positions can lead to positions and employees being rejected for business reasons.
Required fields for new positions are:
EmployeeId
PositionId
PositionStartDate
FirmId
DepartmentId
WorkPercent
(these are required to be able to save the position in WinTid. For the position to be fully usable for an employee there are more fields needed. Which ones depend on the usage of WinTid. See also i) and j)).

e) Positions in the same chain (same CorrelationId on the same employee) can not overlap in time – any change that would cause overlapping positions in the same position chain will be rejected with an error, with one exception explained in f).

f) Any new or updated position that overlaps the latter part of one existing position in the same position chain (i.e. both positions have the same position_correlation_id), will lead to WinTid automatically setting the end date on the previous position to be equal to the day before the start of the newly imported/updated position. Any time data that exists on the previous position inside the date range of the new/updated position will be moved to the new/updated position.

g) If a position has its dates changed so that some existing time data ends up outside the position, that data will be visible in the Ownerless Data page in MinWinTid where it can be transferred to other positions on the employee that currently do not hold time data.

h) If a position has such ownerless data outside the start- and enddates of the position, and a later update adds or changes another position so it now spans this date range AND both the original and the new/updated position have the same CorrelationId, then the time data will be automatically moved to the new position, as long as it does not have time data in the range.

i) New positions added that have an existing earlier position bordering it in the same chain (prior position ends on the day before the new position starts), will have settings copied from the prior position. This includes all settings except position id, start date and end date. This also applies if the imported position has an earlier start date than the end date of the previous position (for example if the previous position had an open-ended start date)

j) New positions added that do NOT have an existing earlier position bordering it in the same chain, will be created with default values if configured. See 7.2 for details.

k) Calendar id, wage group id and fixed overtime id are all settings that include a date, and the setting applies from that date and until it is replaced by a different setting on a later date. If one of these fields is imported WITHOUT a date specified, it applies from the position start date. If an import file attempts to update one of these settings on a date that already has a different id, it will be rejected with an error – but note that calendar has some configurable functionality, see next bullet point.

l) When importing an updated calendar id on a date where a position already has a calendar id, the default action is for the import of that position to fail. This behaviour can be customized by adding an entry in wt_system table, emp_import_calendar_setting which supports the following modes:
    - Default: Same as no setting, same functionality as specified above.
    - Ignore: Any difference in the calendar id in the import data on the same date as an existing calendar id, is ignored – WinTid value will be unchanged, but no error.
    - Overwrite: The imported value will overwrite any existing value on the same date.

m) When importing an updated wage group id on a date where a position already has a wage group id, the default action is for the import of that position to fail. This behaviour can be customized by adding an entry in wt_system table, *emp_import_wagegroup_setting* which supports the following modes:
    - Default: Same as no setting, same functionality as specified above.
    - Ignore: Any difference in the wage group id in the import data on the same date as an existing wage group id, is ignored – WinTid value will be unchanged, but no error.
    - Overwrite: The imported value will overwrite any existing value on the same date.

n) If an existing position has its start date moved back in time, and it had any of calendar, fixedovertime code or wage group id on the original start date – but does NOT have one on the updated startdate, then those settings are moved back in time so they are available from the updated and earlier position start date.

o) Deleting an existing position uses the optional position field “Deleted” set to true. Should it be set on a new position the operation will fail. Should it be used on the only position the operation will also fail due to business reasons. The field does not support any other value and can be omitted if not needed.

NOTE: If some employees fail to import, HTTP 400 Bad Request will be returned, although other employees might have been successfully imported. The response body will contain details on errors.

The post body must contain valid XML or JSON data similar to the other file-based imports. This includes a Header followed by an Employees list of employees, which in turn can contain a list of positions. Some examples included below.

Example json import file importing two new positions and updating the description of a third position on one employee:

```
{
    "EmployeeImport": {
        "Header": {
            "Date": "2021-05-26",
            "Sender": "Agresso",
            "Type": "Employee Import",
        },
        "Employees": [
            {
                "EmployeeId": "3338",
                "Positions": [
                    {
                        "PositionId": "3338-1",
                        "CorrelationId": "main",
                        "DepartmentId": "82",
                        "PositionStartDate": "2020-01-01",
                        "PositionEndDate": "2020-06-30",
                        "WageId": "UnusedWageId",
                        "WorkPercent": "100",
                        "PositionDescription": "First position in chain.",
                        "WageGroupId": "10"
                    },
                    {
                        "PositionId": "3338-2",
                        "CorrelationId": "main",
                        "DepartmentId": "82",
                        "IsPrimaryPosition": "True",
                        "PositionStartDate": "2020-07-01",
                        "WageId": "UnusedWageId",
                        "WorkPercent": "100",
                        "PositionDescription": "second in chain, primary",
                        "WageGroupId": "10"
                    },
                    {
                        "PositionId": "1177",
                        "PositionDescription": "Updated Position Description"
                    }
                ]
            }
        ]
    }
```

Example return values on completion of import

{"status":6,"transactions":1}

See 7.1 for a full list of fields for the Header, Employee and Position entries, as well as a description of the status codes returned.

2.4.4.5 HTTP GET /export/schematimesf

Exports employee working hours per day per employee/position in the specified period. Specifically created to support Success Factors.

Input:

```
{
    "employeeId": 1763,
    "startPeriod": "202009",
    "endPeriod": "202012"
}
```

Note: startPeriod and endPeriod are on the format yyyyMM.

Returns an array with working hours per day in the specified period (truncated example):

```
{
    "WorkScheduleData": [
        {
            "employeeId": "1763",
            "period": "202009",
            "day": 1,
            "hours": 0.0
        },
        {
            "employeeId": "1763",
            "period": "2002009",
            "day": 2,
            "hours": 7.5
        },
        ...
    ]
}
```

NOTE: This endpoint is no longer included in the SWAGGER documentation, since it is an HTTP GET operation with a request body, which does not conform to the rules, and did not validate with https://editor.swagger.io/ In a future version of WinTidAPI this endpoint will change from HTTP GET to HTTP POST to become compliant.

2.4.4.6 HTTP POST /absencessf

Accepts planned absences and sick leaves into the absence handling process integration (full-day absences only)

Absences are added to the absence import queue in *integration_absence_import* table in WinTid, and attempted imported into WinTid as absences. Any that fail will be retained in the integration table with some status information, and can be manually inspected and import retried or rejected via the absence import UI.

```
curl -X POST -H "Content-Type: application/json" -d '{
"ExternalId":777,"EmployeeID":1777, "CorrelationId":"22753",
"StartDate":"20201231", "EndDate":"20201231",
"ExternalAbsenceCode":30, "ApprovalStatus": "Approved", "AbsencePercent":100
}' http://localhost:5000/absencesf
```

Parameter Name	Description
externalId	Unique id generated by the caller (External system)
employeeId	external_employee_id
correlationId	position_correlation_id (string) – which position chain this absence belongs to (Optional – this parameter is not required)
startDate	Start date of absence (yyyyMMdd)
endDate	End date of absence (yyyyMMdd)
externalAbsenceCode	Absence code from the external system will be mapped to WinTid codes via mapping tables integration_Externalsystem and integration_absencecode_in
approvalStatus	Approved / Cancelled (Cancelled = delete)
absencePercent	Percentage of the absence. Should be 100 for most absences, but a partial absence could have a lower value.

In order to use this endpoint, some configuration of winTid is required:

1. Table *integration_externalsystem* must be populated with a system
2. Table *integration_absencecode_in* must be populated with the mapping of absencecodes from the external system and to WinTid absence codes, for the system defined in 1. above.
3. Table *integration_abscodes_delete* can optionally be populated with a list of WinTid absence codes that should be replaced by other imported absences (to support functionality such as replacing employee-registered absence with a sick leave from the external system).
4. WinTid API must have the appSetting key `absenceIntegrationSystemId` set to the system id defined in 1. Above.

- Absences that fail validation can be manually checked in the MinWinTid Absence Import page. If manual resolution of the failure has been done, importing the absence can be retried from this page. Alternatively an absence can be dismissed and will then be archived.
- Absences that do not have a mapping to a WinTid abensecode will fail
- Absences imported, that overlap with other existing absences will fail (unless the existing absence in WinTid has an absence code in *integration_abscodes_delete* and the imported absence completely overlaps it.)
- Absences imported with *approvalStatus* = *CANCELLED* will be removed from WinTid.
- Not using a correlation id at all will attach the absence to the default position chain (correlation id is __default).
- Using a correlation id will attach the absence to the position in a position chain with that correlation id. If a position does not exist in that chain for the absence interval, it will fail.
- If the absence is in a period that has been approved, it will fail.
- Only all-day-absences are currently supported.

Example

```
curl -X POST -H "Content-Type: application/json" -d ' {
    "AbsenceData": [
```

```
        {
            "externalId": "abcf65324b2a47e9add043ea528798ec ",
            "employeeId": "11",
            "startDate": "20190317",
            "endDate": "20190325",
            "absencePercentage": "95",
            "externalAbsenceCode": "NOR_SickWithCertificate",
            "approvalStatus": "APPROVED"
        },
        {
            "externalId": "fg3f65324b2a47e9add043ea528798ec ",
            "employeeId": "7",
            "startDate": "20190301",
            "endDate": "20190303",
            "absencePercentage": "69",
            "externalAbsenceCode": "NOR_UnpaidLeave_F",
            "approvalStatus": "CANCELLED"
           }]}' http://localhost:5000/absencessf
```

2.4.4.7 HTTP GET /employee/lookupbyusername/{username}

Allows looking up the EmployeeId (external_employee_id in Wintid) based on the username.

```
curl http://localhost:5000/employee/lookupbyusername/haveraaenh
```

HTTP/1.1 200 OK

```
{
       "EmployeeId":17524
}
```

```
curl http://localhost:5000/employee/lookupbyusername/nonexisting
```

HTTP/1.1 404 Not Found

```
{
       "Message":"Employee not found",
       "Type":"",
       "Status":"succeeded"
}
```

2.4.4.8 HTTP GET /employeeposition/lookupbypositionid/{positionid}

Allows checking if any position exists that contains the specified external_position_id.

Example looking up by existing external_position_id

```
curl http://localhost:5000/employeeposition/lookupbypositionid/haveraaenh
```

HTTP/1.1 200 OK

```
{
        "Message":"Position found",
        "Type":"",
        "Status":"succeeded"
}
```

Example looking up by non-existing external_position_id

```
curl http://localhost:5000/employeeposition/lookupbypositionid/id_not_used
```

HTTP/1.1 404 Not Found

```
{
        "Message":"Position not found",
        "Type":"",
        "Status":"succeeded"
}
```

2.4.4.9 HTTP POST /addhistoricalselfdeclaredabsence

Allows adding historical self-declared absences to WinTid.

The endpoint supports adding multiple absences for multiple employees in one call.

Example

```
curl -X POST -H "Content-Type: application/json" -d '[
       {"date":"2006-10-10T00:00:00.000Z","employeeId":1777},
       {"date":"2006-10-10T00:00:00.000Z","employeeId":1778},
       {"date":"2006-10-11T00:00:00.000Z","employeeId":1778}
]' http://localhost:5000/addhistoricalselfdeclaredabsence
```

2.4.5 Absencecode module

Allows adding new absence codes to Wintid.

HTTP POST /absencecode

HTTP PUT /absencecode

Input:

```
{
        "Absencecode": 777,
        "Name": "Travel"
}
```

2.4.5.1 Examples

```
curl -X POST -H "Content-Type: application/json" -d '{"Absencecode":777,"Name": "Travel"}'
http://localhost:5000/absencecode
```

2.4.6 Project allocation module

A part of the Messagehub implementation where project allocation is sent to the service bus by MessageHub, and where updates from the service bus are pushed into Wintid by MessageHub. This API is not meant to be accessed directly but is used by MessageHub and is not documented in detail in this document.

HTTP GET /export/projectallocationdays

- lists all project allocations ready for export to MessageHub

HTTP PUT /export/projectallocationdays/status

- updates status on eported project allocation in exp_info table

HTTP PUT/POST /projectallocationday

- accepts project allocation updates from Messagehub and updates WinTid.

HTTP POST /projectallocationday

- accepts project allocation updates from Messagehub and updates WinTid.

HTTP PUT /ep/projectallocationday/approval

- updates and approves project allocation

More detailed messagehub documentation is available upon request.

2.4.7 Organization unit import module

HTTP POST /import/organizationunit

Allows importing new and updated departments.

NOTE: If some departments fail to import, HTTP 400 Bad Request will be returned, although other departments in the same request might have been successfully imported. The response body will contain details on errors.

This endpoint accepts a list of departments to import.

Note: DepartmentIdType is accepted for historical reasons and is not required. N for numeric identifiers on departments, AN for alphanumeric identifiers on departments. This field is not currently used and instead the setting is checked against wt_system value *avdeling_nr_alfa*.

Note: Included fields will be updated. Only the fields necessary for identifying the department are required.

UnitType: What kind of organizational unit. Currently only supports “D” for Department.

UnitId: The numeric or alphanumeric identifier for the department

UnitName: The Department name.

ManagerEmployeeId: The external employee Id of the manager for the department. Must exist in Employee table in WinTid DB.

- If UnitId represents an existing entry, the existing entry is renamed / manager updated.
- If ManagerEmployeeId is an employee in WinTid that is currently not a manager, the import operation will configure the employee as a manager. This requires default values to be enabled and configured since a manager has some required fields that must be filled out, see 7.2 for more information. Importing an employee who is not a currently a manager in wintid, as a manger on a department, will fail if this is not configured.
- A manager added to a department, will also be assigned the department and all employees in the department
- A manager removed or replaced on a department will have the department and any employees in the department that were assigned, removed.

A successful request will return HTTP 200 and the body will contain a Status and list number of Transactions

If any errors occur, HTTP 400 is returned, along with Status, Transactions and an Exceptions array with all errors.

HTTP POST /import/company

Allows importing companies (Firma) into Wintid.

```
curl -X POST -H "Content-Type: application/json" -d '{"Id":777,"Name": "BIM",
"SelfDeclaredSickRuleId":2}' http://localhost:5000/import/company
```

```
{
    "Message": "Company with id 777 imported."
}
```

```
curl -X POST -H "Content-Type: application/json" -d '{"Id":777,"Name": "BOM"
}' http://localhost:5000/import/company
```

```
{
    "Message": "Company with id 777 updated."
}
```

Fields:
Id: Id of the company (firma_nr)
Name: Name of company (firma_navn)
SelfDeclaredSickRuleId: Which self declared sick rule applies to this company (egn_regel)

Status:

Status	Name	Description

0	NotStarted	
1	Parsing	
2	Processing	
3	Finishing	Import has processed all entries, and is now doing cleanup (some like job import might postpone database updates to now)
4	Failed	Import stopped due to failure
5	CompletedWithErrors	Import ran to completion, but one or more records failed to be imported
6	CompletedOk	Import ran to completion and processed all records without errors.

2.4.7.1 Examples

JSON example:

```
curl -v -X POST -H "Content-Type: application/json" -d '@/path/to/example.json'
http://localhost:5000/import/organizationunit
```

Returns: {"Status":6,"Transactions":2}

example.json

```
{
    "OrganizationalUnitImport": {
        "Header": {
            "Date": "2021-01-19",
            "Sender": "WinTid",
            "Type": "Organizational Unit Import",
            "DepartmentIdType": "N"
        },
        "OrganizationUnits": [
            {
                "UnitType": "D",
                "UnitId": "11115",
                "UnitName": "Department 1"
            },
            {
                "UnitType": "D",
                "UnitId": "D3",
                "UnitName": "Department 3",
                "ManagerEmployeeId": "26529"
            }
        ]
    }
}
```

XML example:

```
curl -v -X POST -H "Content-Type: application/xml" -d '@/path/to/example.xml'
http://localhost:5000/import/organizationunit
```

Returns: {"Status":6,"Transactions":2}

example.xml

```
<?xml version="1.0" encoding="utf-8"?>
<OrganizationalUnitImport xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance" xsi:noNamespaceSchemaLocation="OrganizationUnit.xsd" >
  <Header>
    <Date>2008-09-11</Date>
    <Sender>WinTid</Sender>
    <Type>Organizational Unit Import</Type>
    <DepartmentIdType>N</DepartmentIdType>
  </Header>
  <OrganizationUnits>
   <OrganizationUnit>
      <UnitType>D</UnitType>
      <UnitId>11115</UnitId>
      <UnitName>Department 1</UnitName>
    </OrganizationUnit>
    <OrganizationUnit>
      <UnitType>D</UnitType>
      <UnitId>D3</UnitId>
      <UnitName>Department 3</UnitName>
      <ManagerEmployeeId>25629</ManagerEmployeeId>
    </OrganizationUnit>
  </OrganizationUnits>
</OrganizationalUnitImport>
```

2.4.8 Dynamic query module

From version 16.8, WinTid API supports an approach to making customized queries for certain key data elements (initially results – daglig_resultat and related data). For every supported data type, WinTid comes with a default set of fields it returns, and that can be used to filter the requested data.

If additional data is required, a custom view can be created for that type of data, and WinTidApi can be configured to override the default and use the custom view. This is done in appsettings.json for WinTidApi. Only one view can be used at a time of a specific type of data.

Note that the custom view must contain all the fields of the original view, and the added data must be added to the view as CustomField1 up to CustomField15.

When requesting data, filters can be used, and data will only be returned if it matches the values specified in the filter.

Filters can also be applied to the custom fields – but note that they will be parsed as strings.

All filters require an exact match, with the exception of dateFrom and dateTo which matches the date range, inclusive.

By default a query is limited to returning the first 10 000 rows. This can be overridden in the filter by specifying a different number for MaxRows

Note that the performance of this API endpoint will be affected by how the view is constructed and what data is filtered on, so it is important that consideration is made as to how much data will be requested in any single query.

Disclaimer: The default views provided by WinTid should not be changed and should continue working with later versions. Any custom views added by the customer could be broken by changes in later versions of WinTid, and the customer is responsible for keeping them up to date.

2.4.8.1 Daily results

HTTP POST /api/dailyresults/dynamicquery

Returns result data from the daglig_resultat table along with some relevant data from employee and job. The data is returned via the SQL view *wintidapi_result_baseview*.

The view used can be overridden by setting *FlexibleQueryDagligResultatView*to the name of the view to use, in the *WinTidSettings* section of WinTidApi appsettings.json

The available fields are listed below

Field name	Description
absenceTransferred	Absence transfer status of the day of the result
categoryId	Category for the hours of the result
categoryName	Name of the category
correctedResult	Result in hours
correctedResultMin	Result in minutes
correlationId	External identifier for a position chain in WinTid
date	Date of the result
dateFrom	If specified, will return results no earlier than this date
dateTo	If specified, will return results no later than this date

dayApproved	Approval status of the day of the result
departmentId	
externalEmployeeId	Employee identifier from external system
externalPositionId	Position identifier from external system
firmId	
invoiceText	Invoice text, if specified
jobId	Job identifier of the result
jobName	Name of the job
jobStatus	Status of the job (if applicable)
maxRows	The maximum number of rows to return in a call to the api endpoint. Default 10 000
pfield1Id	Project field 1 identifier of the job (if applicable)
pfield2Id	Project field 2 identifier of the job (if applicable)
pfield3Id	Project field 3 identifier of the job (if applicable)
pfield4Id	Project field 4 identifier of the job (if applicable)
pfield5Id	Project field 5 identifier of the job (if applicable)
productionTransferred	Production transfer status of the day of the result
salaryTableId	Salary table on the employee position
salaryTransferred	Salary transfer status of the day of the result

2.4.8.1.1 Curl examples

Returns a maximum of 10 000 rows for Employee 9000 on November 15th 2011:

```
curl -v -X "POST" https://localhost:5000/api/dailyresults/dynamicquery -H "Content-Type: application/json" -d "{\"filter\": {\"date\":\"2011-11-15T00:00:00\", \"externalEmployeeId\":9000}}"
```

Returns a maximum of 100 000 rows for any employee for unapproved days between nov 1st and nov 30th:

```
curl -v -X "POST" https://localhost:44314/api/dailyresults/dynamicquery -H "Content-Type: application/json" -d "{\"filter\": {\"dateFrom\":\"2011-11-01T00:00:00\", \"dateTo\":\"2011-11-30T00:00:00\", \"dayApproved\":0, \"maxRows\":100000}}"
```

2.4.8.1.2 Custom view example

This example creates a view that adds kategori.kategori_kort_navn field as CustomField1. Note that any custom view MUST contain all the original fields, and can add up to 15 custom fields, called CustomField1 through CustomField15:

```
create view wintidapi_result_custom as (
select wintidapi_result_baseview.*, k.kategori_kort_navn as CustomField1 from wintidapi_result_baseview join kategori k on wintidapi_result_baseview.category_id = k.kategori_nr
)
```

Ensure to update WinTidApi appsettings.json to override which view to use:

```
{
    "WinTidSettings": {
        "FlexibleQueryDagligResultatView": "wintidapi_result_custom",
```

Query results will now return CustomField1 containing the kategori_kort_navn field, and queries can be filtered on this field as well:

```
curl -v -X "POST" https://localhost:5000/api/dailyresults/dynamicquery -H "Content-Type: application/json" -d "{\"filter\": { \"dateFrom\":\"2011-11-01T00:00:00\", \"dateTo\":\"2011-11-30T00:00:00\", \"dayApproved\":0, \"customField1\":\"tjen-reis\"}}"
```

2.4.8.2 Employee

HTTP POST /api/employee/dynamicquery

Returns employee data from the employee and employee_position tables along with some relevant department and company data. The data is returned via the SQL view *wintidapi_employee_baseview*.

The view used can be overridden by setting *FlexibleQueryEmployeeView* to the name of the view to use, in the *WinTidSettings* section of WinTidApi appsettings.json

The available fields are listed below

Field name	Description
accessGroupId	
advancedWorkHours	
approveOwnDays	1 if position can approve own days, otherwise 0
birthDate	
canEnterQtyDefects	True if this position can enter quantity and defects, otherwise false.
canFinishJob	True if this position can finish job, otherwise false.
cardNumber	
firmId	
firmName	
costCenterId	
costCenterType	
defaultJobSelection	
deleted	true if the position is deleted, otherwise false. Deleted positions are not included unless set to true in the filter.
departmentId	Internal id of the department
departmentIdAlphanumeric	Internal alphanumeric id of the department (if used)
departmentName	
email	
employeeEndDate	
employeeId	Employee identifier from external system
employeeLastChangeDate	
employeeLastChangedBy	The bruker_id of the manager/superuser who last made changes to the employee.
employeeStartDate	
firstName	
gender	Employee gender (0 – not specified, 1 – male, 2 – female, 3 - other)
hourlyWage	
initials	
jobChoiceMethod	How job search is configured for employee (0 – Job search, 1 – Job search with Pfield Addition, 2 – Pfield search)

lastName	
managerEmployeeId	External employee id of this employee’s manager
maxRows	The maximum number of rows to return in a call to the api endpoint. Default 10 000
maxSelfDeclaredLeave	
minWinTidAccess	True if this position has access to MinWinTid, otherwise false
notificationTypePreference	What kind of push notification preference the employee has (-1 = disabled, 0 = no preference, 1 = web push, 2 = sms, 3 = email)
pfield1Id	
pfield2Id	
pfield3Id	
pfield4Id	
pfield5Id	
phone	
positionCategoryId	
crrelationId	
positionDescription	
positionEndDate	
positionId	Position identifier from external system
positionLastChangeDate	
positionLastChangedBy	
positionStartDate	
primaryPosition	True if the position is tagged as a primary position, otherwise false
projectSetupId	
salaryId	
salaryTableId	
selfDeclaredPeriod	
selfDeclaredRule	
selfDeclaredRuleFromDate	
selfDeclaredSickLeaveLock	
setupId	
sickChildAdjustment	
sickChildCountedInHours	
sickChildDays	
sickChildMaxDays	
sickChildMaxHours	
socialSecurityNumber	
soleCustody	

userName	
vacationAllowanceCurrent	
vacationAllowanceLastYear	
vacationAllowanceOriginal	
vacationAllowanceWithoutPay	
vacationCountedInDays	True if vacation counted in days, false if counted in hours.
vacationRemaining	
vacationTaken	
vacationTransferYear	
vacationWithoutPayBalance	
varExtraDefaultType	
weekWorkHours	
workingPercentage	

2.4.8.2.1 Curl examples

Returns a maximum of 10 000 rows of employees/positions with no vacation days remaining:

```
$ curl -v -X "POST" https://localhost:5000/api/employee/dynamicquery -H "Content-Type: application/json" -d "{\"filter\": {\"vacationRemaining\":0}}"
```

Returns a maximum of 100 000 rows of all employees in the Sales department:

```
$ curl -v -X "POST" https://localhost:5000/api/employee/dynamicquery -H "Content-Type: application/json" -d "{\"filter\": {\"departmentName\":\"Sales\", \"maxRows\":100000}}"
```

2.4.8.2.2 Custom view

*See **Error! Reference source not found.** for example of creating a custom view that contains all the original fields a nd can add up to 15 custom fields.*

Ensure that the database user configured in WintidAPI kunde.config has access to the view.

Ensure to update WinTidApi appsettings.json to override which view to use. Assuming the view is called wintidapi_employee_custom:

```
{
    "WinTidSettings": {
        "FlexibleQueryJobView": "wintidapi_employee_custom",
```

2.4.8.3 Job

HTTP POST /api/job/dynamicquery

Returns data from the jobb table. The data is returned via the SQL view *wintidapi_job_baseview*.

The view used can be overridden by setting *FlexibleQueryJobView* to the name of the view to use, in the *WinTidSettings* section of WinTidApi appsettings.json

The available fields are listed below

Field name	Description
approved	true if the job is approved, otherwise false
budgetHours	
completed	
endDate	Date job was stopped
isDirectJob	Direct job = true, Indirect job (internal) = false
jobbAntall	
jobbProdAntall	
jobId	Unique job identifier
jobName	Name of job
maxRows	The maximum number of rows to return in a call to the api endpoint. Default 10 000
pField1Id	Project field 1 identifier of the job (if applicable)
pField2Id	Project field 2 identifier of the job (if applicable)
pField3Id	Project field 3 identifier of the job (if applicable)
pField4Id	Project field 4 identifier of the job (if applicable)
pField5Id	Project field 5 identifier of the job (if applicable)
plannedCompletionDate	Planned job completion date
plannedStartDate	Planned job start date
responsibleEmployeeId	External employee id of employee responsible for job if any
startDate	Date job was started
status	Status of the job (-20, -10, -1, 0, 2 ,3 ,4)
stopped	true if the job has ended, otherwise false
transferred	true if the job has been transferred to external system, otherwise false

2.4.8.3.1 Curl examples

Returns a maximum of 10 000 rows of jobs with pfield1 set to 817a that have not yet been transferred:

```
$ curl -v -X "POST" https://localhost:5000/api/job/dynamicquery -H "Content-Type: application/json" -d "{\"filter\": {\"pfield1Id\":\"817a\", \"transferred\":0}}"
```

Returns a maximum of 100 000 rows of all approved jobs:

```
$ curl -v -X "POST" https://localhost:44314/api/dailyresults/dynamicquery -H "Content-Type:
application/json" -d "{\"filter\": {\"approved\":1, \"maxRows\":100000}}"
```

2.4.8.3.2 Custom view

*See **Error! Reference source not found.** for example of creating a custom view that contains all the original fields a nd can add up to 15 custom fields.*

Ensure that the database user configured in WintidAPI kunde.config has access to the view.

Ensure to update WinTidApi appsettings.json to override which view to use. Assuming the view is called wintidapi_job_custom:

```
{
    "WinTidSettings": {
        "FlexibleQueryJobView": "wintidapi_job_custom",
```

2.4.8.4 Work Environment Act (AML)

HTTP POST /api/wea/dynamicquery

Returns data from the aml_accumulated_sums table. The data is returned via the SQL view *wintidapi_wea_baseview*.

The view used can be overridden by setting *FlexibleQueryWeaView* to the name of the view to use, in the WinTidSettings section of WinTidApi appsettings.json

The available fields are listed below:

Field name	Description
correlationId	External identifier for a position chain in WinTid
date	Date for the WEA record
dateFrom	If specified, will return WEA records no earlier than this date
dateTo	If specified, will return WEA records no later than this date
departmentId	Id of the employee's department
departmentIdAlpha	Alphanumeric id of the employee's department – if used
departmentName	Department name
employeeId	Employee identifier from external system
firmId	Id of Firm (Company)
firmName	Name of Firm
isFutureSum	0 = This is actual calculated data (and in the past) 1 = This is stipulation of future state, based on the planned work hours for the employee
maxRows	The maximum number of rows to return in a call to the api endpoint. Default 10 000
overtime26Weeks	How many overtime hours the employee has worked in the current 26-week period (assuming the employee overtime is counted in 26 and not 52-week intervals)
overtime4Weeks	How many hours of overtime the employee has worked in the current 4-week period
overtime52Weeks	How many overtime hours the employee has worked in the current 52-week period (assuming the employee overtime is counted in 52 and not 26-week intervals)
overtimeCurrentWeek	How many hours of overtime the employee has worked for the current week
overtimeCurWeekStipulated	Expected overtime hours for the current week, based on planned shifts.
overtimeToday	How much overtime the employee has worked today
positionId	Position identifier from external system
setupId	Id of the WEA setup configured for this employee
violation26WkAvg	1 if the employee on this date has violated their 26-week average working hours, otherwise 0
violation52WkAvg	1 if the employe on this date has violated their 52-week average working hours, otherwise 0
violationAny	1 if the employee on this day is in violation of any work hour or resting time rules, 0 if no rules are violated (and for isFutureSum=1, it is an expectation of a future violation based on current planned shifts)
violationOvertime1Wk	1 if the employee on this day is in violation of their 1-week maximum overtime
violationOvertime26Wk	1 if the employee on this day is in violation of their 26-week maximum overtime

violationOvertime4Wk	1 if the employee on this day is in violation of their 4-week maximum overtime
violationOvertime52Wk	1 if the employee on this day is in violation of their 52-week maximum overtime
violationRestDay	1 if the employee has not had the minimum continuous resting time between shifts
violationRestDayHours	Number of hours violating rest day, 0 if no violation.
violationRestWeek	1 if the employee has worked without allowing for the minimum continuous weekly resting time
violvationRestWeekHours	Number of hours violating rest week, 0 if no violation.
violationWorkhours8WkAvg	1 if the employee has worked too many hours on weekly average, for their current 8-week window on this date.
violationWorkhoursToday	1 if the employee has worked too many hours today, otherwise 0
violationWorkhoursWeek	1 if the employee has worked too many hours this week, otherwise 0
violationWrkhWeekStipulated	1 if the employee is expected to work too many hours for the week on this date, otherwise 0
workhours26WkAvg	How many hours the employee will have worked on average in their current 26-week period, up until this date
workhours26WkAvgStipulated	How many hours the system believes the employee will have worked on weekly average in their current 26-week period, up until this date, based on planned shifts
workhours52WkAvg	How many hours the employee will have worked on average in their current 52-week period, up until this date
workhours52WkAvgStipulated	How many hours the system believes the employee will have worked on weekly average in their current 52-week period, up until this date, based on planned shifts
workhours8WkAvgActual	How many hours the employee has worked on weekly average in the current 8-week window
workhours8WkAvgStipulated	How many hours the system believes the employee will have worked on weekly average in the current 8-week window, based on planned shifts
workhoursCurrentWeek	How many hours the employee has worked this week
workhoursToday	How many hours the employee has worked today

2.4.8.4.1 Curl examples

Returns a maximum of 10 000 rows of wea data for the specified department, in Q1 2024 that have stipulated future wea violations based on expected working hours:

Curl example filtering on department name, with a date range, returning any WEA entries that are stipulated into the future (isFutureSum), with any kind of AML violation (

```
curl -v -X "POST" https://localhost:5000/api/wea/dynamicquery -H "Content-Type: application/json" -d "{\"filter\": {\"departmentName\":\"Maintenance\", \"ViolationAny\":true, \"isFutureSum\":true, \"dateFrom\":\"2024-01-01T00:00:00\", \"dateTo\":\"2024-03-31T00:00:00\"}}"
```

2.4.8.4.2 Custom view

```
{
    "WinTidSettings": {
        "FlexibleQueryWeaView": "wintidapi_wea_custom",
```

2.4.8.5 Registration module

HTTP POST /api/registration/dynamicquery

Returns data from the *stemplinger* table. The data is returned via the SQL view *wintidapi_registration_baseview*.

The view used can be overridden by setting *FlexibleQueryRegistrationView* to the name of the view to use, in the WinTidSettings section of WinTidApi appsettings.json

The available fields are listed below:

Field name	Description
absenceCode	Absence code of the registration (if any)
coreTime	1 if registration is within the Core time
correlationId	External identifier for a position chain in WinTid
costCenterId	Cost center id (for change cost center registrations)
created	When the registration record was added to the db
date	Date of the registration
dateConsidered	Date the registration applies to (could be the day before or after the actual registration)
defects	Number of defects
departmentId	The id of the department of the position making the registration
departmentIdAlpha	The Alphanumeric department id (if used)
departmentName	The name of the department of the position making the registration
employeeId	WinTid internal id of the employee.
externalEmployeeId	Employee identifier from external system
externalPositionId	Position identifier from external system
extraId	Extra id (for extra registrations)
extraValue	Extra value (for extra registrations)
firmId	Id of the firm the position that made the registration belongs to
firmName	Name of the firm the position that made the registration belongs to
firstName	Employee first name
jobId	Id of the job this registration applies to (if any)
lastName	Employee last name
overtimeCode	Overtime code of the registration (if any)
pfield1	
pfield2	
pfield3	
pfield4	
pfield5	

positionId	WinTid internal id of the position
quantity	Number of produced items (for job registrations)
registrationCode	Currently not in use
registrationDate	The date the registration was done
registrationDateConsidered	The calculated date of the registration
registrationDateTime	Date and time of the registration
registrationErrorCode	Currently not in use
registrationFlag	1 = In due to wrong sequence 2 = calculated out due to forgotten out registration 3 = Currently not used 4 = Currently not used 5 = In due to planned absence start 6 = Out due to planned absence end 7 = Currently not used 8 = Adjustment due to overtime code 9 = Adjustment due to past midnight registration 10 = Automatic in registration (calculated)
registrationFlagName	Norwegian description of the registration flag
registrationReason	The specified reason for the registration (if any)
registrationTimeConsidered	Time for registration after calculation (due to f ex rounding)
registrationType	Technical KABA name for registration (e.g. B1, B2, FE)
registrationTypeName	Norwegian description of the registration type
registrationVersion	0= Original version. >0 = registration correction id
returnDate	Currently not in use
returnTime	Currently not in use
sequenceNumber	Unique sequence number for the registration
terminalAddress	The KABA address (gid/did) of the terminal where the registration originated eg @A (note, some internal identifiers also exist that start with X and do not identify a KABA terminal)
terminalLocation	An optional description of where the terminal is located
time	Time of registration
userId	User id of a manager/superuser who added/edited a registration

2.4.8.5.1 Curl examples

Returns a maximum of 10 000 rows of registrations for the specified department, in Q1 2024 that have stipulated future wea violations based on expected working hours:

Curl example retrieving all Inn registrations for employee with external employee id 104:

```
curl -v -X "POST" https://localhost:5000/api/registration/dynamicquery -H "Content-Type:
application/json" -d "{\"filter\": {\"externalEmployeeId\":104, \"registrationTypeName\":\"Inn\"}}"
```

2.4.8.5.2 Custom view

```
{
    "WinTidSettings": {
        "FlexibleQueryRegistrationView": "wintidapi_registration_custom",
```

2.4.9 Export to API module

HTTP POST/PUT /exportToApi/UpdateRecordReceipts

Allows updating the status, external error code and external error messages of existing records within the api_export_absence_data and api_export_various_data tables.

This endpoint accepts a list of records to update.

NOTE: The operation will fail if any of the records are not of a various or an absence export type.

NOTE: The operation will fail if any of the records have provided an error code that is different from 0 but an error message is missing.

NOTE: The operation will fail if any of the records have provided invalid RowIds which don't match the ones in the database.

Fields:

- **ExportType**: What kind of a record is this. Any other value will result in an error.

ExportType	Name	Description
0	Mustering	Various export
1	Cost Center	Various export
2	Absence	Absence export
3	Various - Production	Various export

- **RowSendingId:** The numeric (integer) identifier for the record to update. If this row id is not found in the database an error will be thrown.
- **ErrorCode:** The numeric (integer) error code. 0 if the record was updated fine and no error exists. Any other value also requires a non-empty ErrorMessage and will set the status of this record to be Rejected by the external system.
- **ErrorMessage:** The string message of the reason for the error with a maximum limit of 512 characters. This will be shown in the Failed Transactions Overview page to the user.

A successful request will return HTTP 200. If any errors occur, HTTP 500 is returned along with the Status and the Exception.

Refer to Swagger for any examples regarding this endpoint.

3 File based imports

Wintid supports scheduled and ad-hoc import of new and updated data from files for a number of central entities in the system, including:

- Employees
- Departments
- Jobs / Projects
- Pfields
- Result corrections
- Inndata (relevant when using physical KABA terminals)

This chapter details the different file-based import types.

3.1 Automating file-based imports

Wintid includes a WintidIntegrationService, a Windows service that is configured in Wintid. It can be set to automatically import all files of the specific import type from a source folder.

3.2 Import file formats and header

All imports support XML format as defined by the XML schema files provided with Wintid (for example EmployeeSchema.xsd). All import files should have the same prefix as the entitytype, for example EmployeeImportDec2020.xml, and must pass XML validation against the XML schema for that import type.

Every XML import file begins with a header, that contains the following fields

Header field name	Description
Date	When the import applies. This can be important for example when importing updates that apply on a specific date, for example an updated calendar assignment to an employee, or a change that creates a new position. *If Date is not specified, the date of the import is assumed.*
Sender	Can be used to indicate where the import data comes from

Type	Must be set to the correct string based on the import type, or import will fail:
	`Employee Import`
	`Organizational Unit Import`
	`Balance Import`
	`Pfelt Import`
	`Job Import`

DepartmentType	Indicates whether departments are identified by numeric id or by alphanumeric id. Legal values are: `AN` - Alphanumeric `N` - Numeric

3.3 Employee import

Allows importing new and updated employees. For deleting employees, a resignation date can be imported, but employees cannot be directly deleted using employee import. Wintid schedules a job to delete resigned employees after a configurable retention period. Note that one import file can contain many employees

See EmployeeSchema.xsd for details.

3.3.1 Example import file

Many fields are optional, so an actual import file will often be far smaller in scope. The below example sets the end date and updates the Pfield1 filter on employee with ExternalEmployeeId 1673.

```
<?xml version="1.0" encoding="iso-8859-1"?>
<EmployeeImport xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="EmployeeSchema.xsd" >
  <Header>
    <Date>2020-10-16</Date>
    <Sender>Excel</Sender>
    <Type>Employee Import</Type>
    <DepartmentType>N</DepartmentType>
  </Header>
  <Employees>
    <Employee>
      <EmployeeImportType>Change</EmployeeImportType>
      <EmployeeId>1673</EmployeeId>
      <EngageToDate>2021-01-01</EngageToDate>
      <Pfield1>5</Pfield1>
    </Employee>
  </Employees>
</EmployeeImport>
```

3.4 Organizational unit import

Allows importing new and updated departments into the Avdeling table in Wintid.Currently only allows importing departments, which means the UnitType *must* be “D” in the importfile.

See OrganizationUnit.xsd for details.

3.4.1 Example import file

The below file imports 2 departments.

```
<?xml version="1.0" encoding="utf-8"?>
<OrganizationalUnitImport xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="OrganizationUnit.xsd" >
  <Header>
    <Date>2020-09-11</Date>
    <Sender>WinTid</Sender>
    <Type>Organizational Unit Import</Type>
    <DepartmentIdType>AN</DepartmentIdType>
  </Header>
  <OrganizationUnits>
    <OrganizationUnit>
      <UnitType>D</UnitType>
      <UnitId>D1x</UnitId>
      <UnitName>Department 1</UnitName>
    </OrganizationUnit>
    <OrganizationUnit>
```

```
      <UnitType>D</UnitType>
      <UnitId>D2x</UnitId>
      <UnitName>Department 2</UnitName>
    </OrganizationUnit>
  </OrganizationUnits>
</OrganizationalUnitImport>
```

3.5 Job / Project import

Allows importing jobs/projects/cost centers into Wintid jobb table.

Note that new pfields specified in on a job in the import file, or pfields with updated names, will also update those pfields in the respective pfeltN table in Wintid – as long as Wintid is configured to allow this.

See JobSchema.xsd for details on the format of the job XML.

NOTE: The Job import also supports importing jobs in CSV format.

3.5.1 Example import file

```
<?xml version="1.0" encoding="utf-8"?>
<JobImport xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="JobSchema.xsd" >
  <Header>
    <Date>2020-09-11</Date>
    <Sender>Agresso</Sender>
    <Type>Job Import</Type>
  </Header>
  <Jobs>
    <Job>
      <JobStatus>-1</JobStatus>
      <JobId></JobId>
      <JobName> Integration-Design </JobName>
      <Pfield1No>111</Pfield1No>
      <Pfield1Name>SupportTeam</Pfield1Name>
      <Pfield2No>222</Pfield2No>
      <Pfield2Name>ReleaseTeam</Pfield2Name>
      <Pfield3No></Pfield3No>
      <Pfield3Name></Pfield3Name>
      <Pfield4No>111</Pfield4No>
      <Pfield4Name>DyNamicsteam</Pfield4Name>
      <Pfield5No></Pfield5No>
      <Pfield5Name></Pfield5Name>
      <JobBudget>11,5000</JobBudget>
      <JobPlannedStartDate>2020-09-11</JobPlannedStartDate >
      <JobPlannedEndDate>2022-01-01</JobPlannedEndDate>
      <JobCardPrintedOut>0</JobCardPrintedOut >
      <JobCompleted>0</JobCompleted>
      <Finished>0</Finished>
      <Approved>0</Approved>
      <Transferred>0</Transferred>
    </Job>
  </Jobs>
</JobImport>
```

Note: Job import also supports import via CSV format, as in the below example (copy into a separate file for better viewing.):

```
JobStatus;JobId;JobName;Pfield1No;Pfield1Name;Pfield2No;Pfield2Name;Pfield3No;Pfield3Name;Pfield4No;Pfield4Name;Pfield5No;Pfield5Name;JobBudget;JobPlannedStartDate;JobPlannedEndDate;JobCardPrintedOut;JobCompleted;Finished;Approved;Transferred
-1;; Integration-Design ;111;SupportTeam;222;ReleaseTeam;;;111;Dynamicsteam;;;11,5000;2009-07-30;2009-06-30;0;1;1;1;1
-1;; Integration-Design ;111;SupportTeam;777;Tester;;;111;Dynamicsteam;;;11,5000;2009-07-30;2009-06-30;0;1;1;1;1
```

3.6 Pfield import

Allows import of new and updated Pfields to the respective pfeltN tables in Wintid.

See PFieldSchema.xsd for details.

3.6.1 Example import file

The below import file imports one pfield at each level 1-5.

```
<?xml version="1.0" encoding="utf-8"?>
<PfeltImport xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="PfeltUnit.xsd" >
  <Header>
    <Date>2008-09-11</Date>
    <Sender>WinTid</Sender>
    <Type>Pfelt Import</Type>
  </Header>
  <PfeltUnits>
    <PfeltUnit>
      <PfeltType>1</PfeltType>
      <PfeltId>8101</PfeltId>
      <PfeltName>Pfelt11</PfeltName>
    </PfeltUnit>
    <PfeltUnit>
      <PfeltType>2</PfeltType>
      <PfeltId>8201</PfeltId>
      <PfeltName>Pfelt22</PfeltName>
    </PfeltUnit>
    <PfeltUnit>
      <PfeltType>3</PfeltType>
      <PfeltId>8301</PfeltId>
      <PfeltName>Pfelt33</PfeltName>
    </PfeltUnit>
    <PfeltUnit>
      <PfeltType>4</PfeltType>
      <PfeltId>8401</PfeltId>
      <PfeltName>Pfelt44</PfeltName>
    </PfeltUnit>
    <PfeltUnit>
      <PfeltType>5</PfeltType>
      <PfeltId>8501</PfeltId>
      <PfeltName>Pfelt55</PfeltName>
    </PfeltUnit>
    <PfeltUnit>
      <PfeltType>5</PfeltType>
      <PfeltId>8501</PfeltId>
      <PfeltName>Pfelt5</PfeltName>
    </PfeltUnit>
```

```
  </PfeltUnits>
</PfeltImport>
```

3.7 Balance import

Allows overriding the day/hour balances for an employee on a given category and date.

See BalanceSchema.xsd for details.

3.7.1 Example import file

The below example file shows updating the balance on category 2 for employee with ExternalEmployeeId 1672 on May 18th 2020, setting a new balance to 10.4 hours.

```
<?xml version="1.0" encoding="utf-8"?>
<BalanceImport xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="BalanceSchema.xsd">
  <Header>
    <Date>2008-11-11</Date>
    <Sender>SAP</Sender>
    <Type>Balance Import</Type>
  </Header>
  <ResultCorrections>
    <ResultCorrection>
      <EmployeeId>1672</EmployeeId>
      <Date>2020-05-18</Date>
      <CategoryNo>2</CategoryNo>
      <Balance>10.4</Balance>
      <CorrectionType>3</CorrectionType>
    </ResultCorrection>
  </ResultCorrections>
</BalanceImport>
```

4 File based exports

File based exports are mainly out of scope for this document, but an overview is included for completeness.

Wintid has flexible support for custom export formats for file-based exports of the following central Wintid data:

- Results / hours
- Absences
- Working hours

Custom export file formats can be defined for common formats like fixed-length, CSV and XML, using UTF-8 or iso-8859-1 encoding.

Exports can be scheduled to run automatically.

For more details on managing exports from Wintid, please see Wintid brukerdokumentasjon.

Note: Most exports will run in a parallelized fashion, utilizing all logical CPU cores on the machine. In some scenarios this might put too high a load on either the application server or the database server. The degree of parallelization can be adjusted by adding the *parallellisering* parameter to the kunde.config file of the application in question:

```
<WebApplikasjonsInnstillinger
type="Ementor.Medina.ApplikasjonsInnstillinger.WebApplikasjonsInnstillinger, Ementor.Medina,
Culture=neutral"
 parallellisering="3"

...

/>
```

5 API based exports

5.1 Introduction

Wintid supports export of mustering, production, cost center and absence data to an external API. This follows the same broad process as a file-based export, except that instead of exporting the data to a file, it is exported to internal export tables, and from there will be posted to the configured external API.

The API based exports also supports a receipt mechanism, where an external system can call back to a receipt endpoint provided by WinTid API that allows the external system to mark any transaction as *rejected* or *accepted*.

Rejected transactions will lead to an email sent out to the configured manager for the affected employee(s) according to the configured WinTid Scheduler job schedule. These transactions will be available to the manager for review in the Failed Transaction Overview page, where the manager can see the reasons for the failure, and either dismiss the transaction(s) or attempt to re-send them after manually correcting the cause.

5.2 Formats

The API-based export currently supports a JSON format that is defined in the Export Configuration page. The supported formats are the same for file exports and export to api, with the one difference that an export to API can be split into multiple calls.

Note: If the receipt API is to be used to reject/accept transactions, then the format needs to include the *row sending id* so it is available to the external system, as this id is used to identify the specific transaction to reject or accept.

Simple example json format:

```
[
    {
        "rowSendingId": "784",
        "employeeId": "1234567",
        "date": "2022-03-04",
        "wagetype": "121",
        "result": "7.00"
    },
    {
        "rowSendingId": "785",
        "employeeId": "1234568",
        "date": "2022-03-04",
        "wagetype": "121",
        "result": "2.50"
    }
]
```

5.3 Process

When an export to API is executed, either via the Manual Export page or scheduled by Wintid Integration Server, the data is retrieved from WinTid, written to export tables, and usually configured to be marked as transferred. For all

intents and purposes the data has now been exported from WinTid. In the export tables, any added rows will have the status *new.*

Immediately after the data has been exported, it will be attempted posted to the external API. Depending upon the api configuration in WinTid this could be a single post of all transaction or several calls – up to one call per transaction. All rows that are successfully posted will have their status changed to *sent* in the export tables. Any rows that fail to be posted for whatever reason, will retain their status as *new*.

If the Export Configuration is set up to mark rows as transferred, re-running the export will not add any new rows to the export tables (unless there are new rows, for example if the export selection criteria are changed). However, all new and existing rows connected to the same Export Setup with the status *new* will be attempted sent. This allows rows that were unable to be sent earlier due to transient errors, to be sent at the next opportunity.

5.4 Api configuration

API based exports are enabled by creating a JSON format export that is configured to *transfer via API* and by adding the API-specific configuration settings of *URL,* Authentication settings and how many rows to export in each call to the external API (set to 0 to indicate that all rows can be sent in a single call).

The currently supported authentication schemes is HTTP Basic authentication and JWT Bearer authentication.

HTTP Basic authentication adds an Authorization header to all the calls:

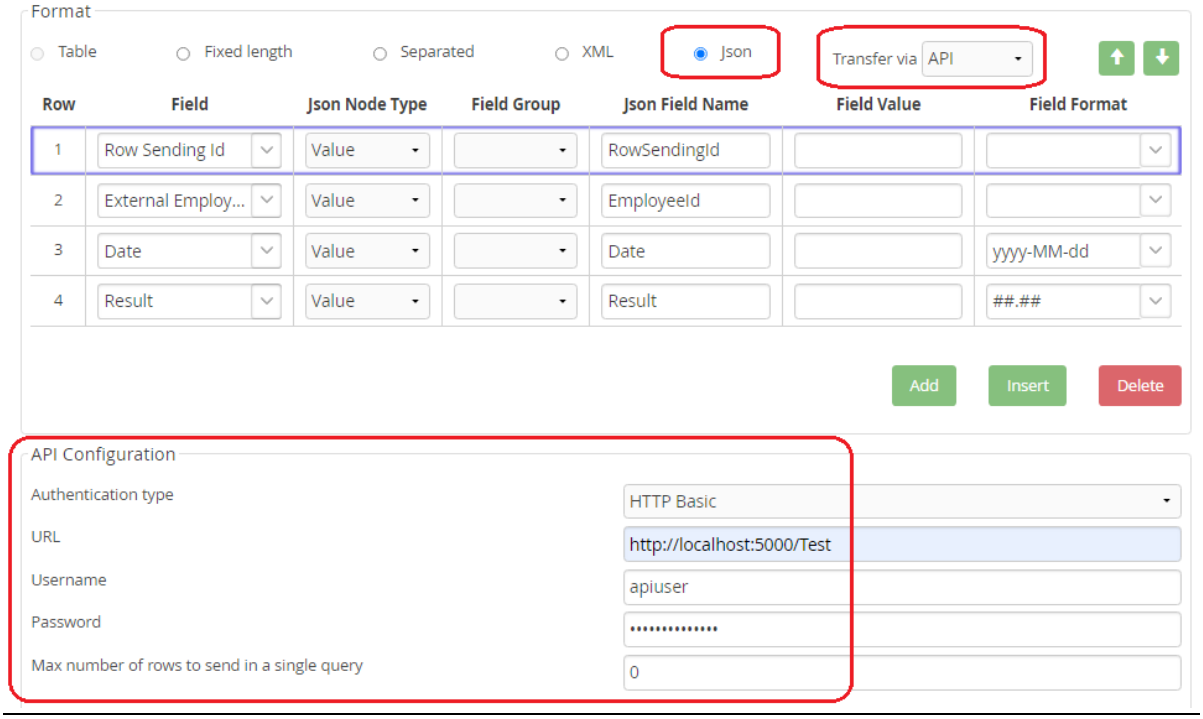
Authorization: Basic YXBpdXNlcjpzb21lbWFnaWNwYXNzd29yZA==

Where the specified username and password are concatenated with a colon and base-64 encoded.

JWT Bearer authentication adds the following header:

Authorization: Bearer EhZxD6p64LcNDnxB

Where the specified bearer token is used.



5.5 Receipt endpoint

The receipt endpoint is documented under WinTid API in 2.4.8

By providing the *row sending id* of a transaction that has been successfully posted by WinTid to the external API, any exported transaction can be marked in the WinTid export tables as *accepted* or *rejected*. If it is marked as a rejected, the caller must also provide an error code and error message, that will be available for review in the Failed Transaction Overview Page in WinTid.

For marking a transaction *accepted*, no more information is required.

The receipt endpoint expects a json array with one or more receipts – it is fine to batch receipts for performance reasons.

5.6 Notes on the row sending id

The *row sending id* is a unique id per *export type*. This means that the same id can appear both in a mustering export and an absence export for example.

Use the *row sending id* along with the export type to uniquely identify a transaction to accept/reject.

After a manager has reviewed failed transactions in the Failed Transaction Overview Page and attempts to re-send them, they will be sent with a **new** *row sending id*, and any further accept/reject calls to the Receipt Endpoint must use this new id to accept/reject the transaction.

The *row sending id* is only set by the API export process, so if an export is configured to include the *row sending id* and export to a file, then it will only contain the value -1

6 WinTid Integrationservice

Note: This service is not actively being developed and will be deprecated in a later version of WinTid.

Wintid Integrationservice is an IIS-hosted Soap 1.1 compatible web service that provides a number of integration points with Wintid.

The integration points can be explored from a browser by opening /WintidIntegrationServer.asmx

Additionally, it is self-describing using WSDL that can be reached at /WintidIntegrationServer.asmx?WSDL and allows a number of modern programming languages to automatically create plumbing code to access the functionality (for example by adding a service reference in a Visual Studio .net project)

6.1 Integration points

6.1.1 DoFileImport

Allows the triggering of a pre-configured file import by an external service

6.1.2 GetDepartmentInfo

Returns a list of all departments

6.1.3 GetEmployeeInfoAll

Returns a list of every position in the database (note that employees with multiple positions could appear multiple times)

6.1.4 GetEmployeeInfoInDepartment

Returns a list of every position in the specified department (note that employees with multiple positions could appear multiple times)

6.1.5 GetEmployeeInfoInFirm

Returns a list of every position in the specified Firm (note that employees with multiple positions could appear multiple times)

6.1.6 GetFirmInfo

Returns a list of all Firms defined in Wintid

6.1.7 GetHistoricalAbsence

Given a date range and list of ExternalEmployeeIds, returns one entry for each employee/date/category of absence from daglig_resultat table.

6.1.8 GetResultsInHours

Given a date range, a list of ExternalEmployeeIds and an approval level, returns one entry for each employee/date/category of results from daglig_resultat where the approval level is at least as high as requested.

6.1.9 GetScheduledAbsence

Returns a list of planned absences for the specified positions in the specified date range. This includes both data from planlagt_fravaer as well as ansatt_avvik in Wintid.

6.1.10 GetSchemaInfo

Returns basic information on all calculation schemes in Wintid, including number of working hours, start and end of core working time and absence times.

6.1.11 GetSchemaInfoBySchemaId

Returns basic information on the specified calculation scheme, including number of working hours, start and end of core working time and absence times.

6.1.12 GetSchemaTime

Returns information on which schema is used by all requested positions on all requested days.

6.1.13 ImportProjectResult

Imports a new or updated project result on the specified position id, date, category and job/project.

6.1.14 UpdateSchemaTime

Updates which calculation scheme an employee position is assigned to on a specific date.

7 References

7.1 Employee import field overview

7.1.1 Header fields

Import field	Description
Date	Date for when the imported data applies. This is relevant since some of the imported data will be tagged with this date, known as “Header date”. If not set, uses the current date at the time of the import. Format: YYYY-MM-DD
Sender	Information string of the source of data
Type	Type of import. Should be set to “Employee Import”
DepartmentType	Defaults to “N” for numeric. Set to “AN” if this WinTid installation uses alphanumeric department identifiers.

7.1.2 Employee fields

Import field	Table	Table column	Notes
DateOfBirth	Employee	birth_date	Format: YYYY-MM-DD
Email	Employee	email	Supports a comma-separated list of emails, up to 512 characters e.g. “on@test.com, ola.nordmann@test.com”
EmployeeId	Employee	external_employee_id	
EmployeeImportType			Not used by the WinTidAPI imports
EngageFromDate	Employee	employee_start_date	
EngageToDate	Employee	employee_end_date	Set to 2000-01-01 to change end date to open-ended
EntitleHoliday	Employee	vacation_allowance_current	
FirstName	Employee	first_name	
Initials	Employee	Initials	
LastName	Employee	last_name	
NewEmployeeId	employee	external_employee_id	Set this to change the external_employee_id of the employee.
optionalTexts	ansatt_fritekst	ansatt_fritekst_tekst	"optionalTexts":[{"textID":"pfelt1","textValue":"Unit"}, {"textID":"pfelt2","textValue":"Test"}] Can contain an array/list of names and values. Note that the textID is the dbo.fritekst.fritekst_id, not fritekst_navn.
OriginalHolidayValue	Employee	vacation_allowance_original	
Password	Employee	password	
Sex	Employee	gender	

SocialSecurityNumber	Employee	social_security_number	
Phone	Employee	phone	

7.1.3 Position fields

Import field	Table	Table column	Notes
ApplicationSetupId	Employee_Position	setup_id	
CalendarFromDate	ansatt_kalender	kalender_fra_dato	
CalendarId	ansatt_kalender	kalender_nr	Adds the specified calendar id on the date specified by CalendarFromDate, or position start date if CalendarFromDate is not specified.
CardId	Employee_Position	card_number	
CorrelationId	employee_Position	position_correlation_id	If set, indicates that the position is in a chain of connected positions, with all other positions on the same employee that share the same correlationId
CostCenterFromDate	ansatt_ksted	ksted_fradato	If Cost Center Levels are specified, uses this date to configure cost centers
CostCenterLevel1	ansatt_ksted	ksted_nr	This value along with the company setting on the position is used to set the correct cost center
CostCenterLevel2	ansatt_ksted	ksted_nr	This value along with the company setting on the position is used to set the correct cost center
CostCenterLevel3	ansatt_ksted	ksted_nr	This value along with the company setting on the position is used to set the correct cost center
CostCenterLevel4	ansatt_ksted	ksted_nr	This value along with the company setting on the position is used to set the correct cost center
CostCenterLevel5	ansatt_ksted	ksted_nr	This value along with the company setting on the position is used to set the correct cost center
Deleted	employee_position	deleted	Marks the position as deleted if this field is set to true. Is optional and can be omitted, false value is disregarded. Note: only supported in the 2.4.4.3 endpoint at import/employeeandposition
DepartmentId	Employee_Position	department_id	
FirmId	Employee_Position	company_id	
FixedOvertimeFromDate	Fast_overtidskode	fast_overtidskode_fra_dato	
FixedOvertimeId	Fast_overtidskode	overtidskode_nr	Adds the specified FixedOvertimeId to the date specified by FixedOvertimeFromDate, or position start date if FixedOvertimeFromDate is not specified.
IsPrimaryPosition	Employee_Position	primary_position	
ManagerEmployeeId	Employee_Position	manager_employee_id	Set this to the External_employee_id of the manager. Will set the correct employee_id on the position.
Pfield1	Employee_Position	pfield1_id	
Pfield1Filter	employee_pos_pfield_filter	pfield_filter	

Pfield2	Employee_Position	pfield2_id	
Pfield2Filter	employee_pos_pfield_filter	pfield_filter	
Pfield3	Employee_Position	pfield3_id	
Pfield3Filter	employee_pos_pfield_filter	pfield_filter	
Pfield4	Employee_Position	pfield4_id	
Pfield4Filter	employee_pos_pfield_filter	pfield_filter	
Pfield5	Employee_Position	pfield5_id	
Pfield5Filter	employee_pos_pfield_filter	pfield_filter	
PositionCategoryId	Employee_Position	position_category_id	
PositionDescription	Employee_Position	description	
PositionEndDate	Employee_Position	position_end_date	
PositionId	Employee_Position	external_position_id	
PositionStartDate	Employee_Position	position_start_date	
ProjectSetupId	Employee_Position	project_setup_id	
WageGroupFromDate	Ansatt_lonnsgruppe	ansatt_lonnsgruppe_fra_dato	
WageGroupId	Ansatt_lonnsgruppe	lonnsgruppe_nr	Adds the specified WageGroupId on the date specified in WageGroupFromDate, or on position start date if WageGroupFromDate is missing.
WageId	Employee_Position	salary_id	
WageTypeTableId	Employee_Position	salary_table_id	
WorkPercent	Employee_Position	working_percentage	
WeekWorkHours	Employee_Position	week_work_hours	
HourlyWage	Employee_Position	hourly_wage	

7.1.4 Status codes

Code	Name	Description
0	NotStarted	
1	Parsing	
2	Processing	
3	Finishing	Import has processed all entries, and is now doing cleanup (some like job import might postpone database updates until this point)
4	Failed	Import stopped due to failure
5	CompletedWithErrors	Import ran to completion, but one or more records failed to be imported
6	CompletedOk	Import ran to completion and processed all records without errors

7.1.5 Import example

```
curl --location --request POST 'http://localhost:81/import/employees' \
--header 'Content-Type: application/json' \
--data-raw '{
    "EmployeeImport": {
        "Header": {
            "Date": "2021-08-30",
            "Sender": "Agresso",
            "Type": "Employee Import",
            "DepartmentType": "N"
        },
        "Employees": [
            {
                "EmployeeImportType": "New",
                "EmployeeId": "1701",
                "PositionId": "1701_1",
                "LastName": "Some",
                "FirstName": "Body",
                "Sex":"K",
                "DateOfBirth":"1971-08-23",
                "EngageFromDate":"2009-08-01",
                "UserName":"somebody",
                "Password":"wintid123",
                "Email":"somebody@wintid.no",
                "PositionDescription": "Første_stilling",
                "PositionStartDate": "2021-09-03",
                "PositionEndDate": "2099-12-31",
                "FirmId": "100",
                "DepartmentId": "20",
                "ApplicationSetupId": "64",
                "CardId": "7000",
                "PositionCategoryId": "0",
                "WageId": "7000",
                "WageTypeTableId": "5",
                "WorkPercent": "100",
                "IsPrimaryPosition": "True",
                "ManagerEmployeeId": "1610",
                "ProjectSetupId": "1",
                "CalendarId": "998",
                "CalendarFromDate": "2021-09-03",
                "FixedOvertimeId": "-1",
                "WageGroupId": "10",
                "CostCenterFromDate": "2021-03-09",
                "CostCenterLevel1": "0017",

                "Pfield1": "S_1001",
                "Pfield1Filter": "S_*",
                "Pfield2": "205",
                "Pfield2Filter": "2*"
            }
```

```
        ]
    }
}
'
```

7.2 Default values

To import employees *who are currently not managers* in WinTid as a manager on a department, default values must be enabled.

Enabling default values consists of two steps:

Configure the path to the folder containing default values
In WinTidApi this is done by adding a WinTidDefaultsDir entry in the WinTidSettings section of appsettings.json:

```
"WinTidSettings": {
   "AbsenceIntegrationSystemId": "1",
   "WinTidDefaultsDir": "C:\wintid\defaultvalues"
},
```

Place a json file in the folder defined above. The filename must begin with *person* e.g. persondefaults.json. The file contents are similar to the format of a snapshot import json, with a single employee and that employee having a single position entry. All fields available in snapshot import can be configured with a default value that is applied when a new employee or new position is created by snapshot import. Note that any fields supplied in the import file will override any default setting.

Additionally, the employee can have a *Manager* section to define default settings that will be applied when organization unit import is used to set an employee as a manager on a department, when that employee is not yet configured as a manager in WinTid. At a minimum, the Manager section must include an ApplicationSetupId and a Usergroup to apply to any new managers, as these are required fields. The complete list of manager fields that can be set:

- ApplicationSetupId
- Usergroup
- ApproveCostCenter
- ApproveProject
- UnaproveTransferred
- FullSelfAccess
- OverrideActivityAllocation

Example person.json:

```
{
    "EmployeeImport": {
        "Header": {
            "Date": "2021-10-30",
            "Sender": "Agresso",
            "Type": "Employee Import",
            "DepartmentType": "N"
```

```
        },
        "Employees": [
            {
                "EntitleHoliday": "25",
                "OriginalHolidayValue": "25",
                    "Positions": [
                        {
                        "FirmId": "2",
                        "ApplicationSetupId": "64",
                        "WorkPercent": "100",
                        "CalendarId": "300",
                        "FixedOvertimeId": "1",
                        "WageGroupId": "30",
                        "Pfield1": "S_1001",
                        "Pfield1Filter": "*",
                        "Pfield2": "206"
                        }
                        ],
                "Manager": {
                    "ApplicationSetupId": "83",
                    "Usergroup": "1",
                    "ApproveCostCenter": "True",
                    "ApproveProject": "True",
                    "UnaproveTransferred": "true",
                    "FullSelfAccess": "true",
                    "OverrideActivityAllocation": "true"
                }
            }
        ]
    }
}
```

7.3 Job import field overview

Import field	Table	Table column	Notes
Approved	jobb	godkjent	1 - job is approved, 0 - job is not approved
div1			not used / customer specific
JobBudget	jobb	jobb_budsjett	job budget in hours
JobCardPrintedOut	jobb	jobb_kort_utskrevet	
JobCompleted	jobb	avsluttet	1 - job is completed, 0 - job is not completed
	jobb	jobb_nr	unique job id
JobName	jobb	jobb_navn	unique job name
JobNumber	jobb	jobb_antall	quantity produced
JobPlannedEndDate	jobb	jobb_plan_slutt	
JobPlannedStartDate	jobb	jobb_plan_start	
JobProducedNumber	jobb	jobb_prod_antall	
JobStatus	jobb	jobb_status	Job status: -1: Planned (not started) 0: Ongoing 1: Stopped 2: Completed 3: Totally Finished 4: Discontinued / Expired
Pfield1Name	pfelt1	pfelt1_navn	The Pfield name of a new or existing pfield
Pfield1No	jobb, pfelt1	pfelt1_nr	The pfield number of a new or existing pfield
Pfield2Name	pfelt2	pfelt2_navn	The Pfield name of a new or existing pfield
Pfield2No	jobb, pfelt2	pfelt2_nr	The pfield number of a new or existing pfield
Pfield3Name	pfelt3	pfelt3_navn	The Pfield name of a new or existing pfield
Pfield3No	jobb, pfelt3	pfelt3_nr	The pfield number of a new or existing pfield

Pfield4Name	pfelt4	pfelt4_navn	The Pfield name of a new or existing pfield
Pfield4No	jobb, pfelt4	pfelt4_nr	The pfield number of a new or existing pfield
Pfield5Name	pfelt5	pfelt5_navn	The Pfield name of a new or existing pfield
Pfield5No	jobb, pfelt5	pfelt5_nr	The pfield number of a new or existing pfield
ResponsibleEmployeeId	jobb	responsible_employee_id	The unique external employee id of the employee responsible for the job.
Transferred	jobb	overfort	1 - job has been transferred, 0 - job has not been transferred
Approved	jobb	godkjent	1 - job is approved, 0 - job is not approved

REVISION HISTORY

Version	Date	Author	Version	Description
1.0	15.12.2020	IS		Combined several documents into this document
2.0	19.01.2021	IS		Added API for Organization unit import module
2.1	19.02.2021	IS		Updates for new and updated endpoints
2.2	16.03.2021	IS		Look up position by external position id
2.3	28.04.2021	IS		Absence integration endpoint
2.4	29.06.2021	IS		Employee position snapshot import
2.5	29.08.2021	IS		Employee position section updated, added references at end
2.6	11.11.2021	IS		15.1 updates to Employee position snapshot import, org. unit import and default values
2.6.1	18.11.2021	IS		Added parallellisering parameter to file exports.
2.6.2	30.11.2021	IS	15.2	WinTidAPI new endpoint addhistoricalselfdeclaredabsence Support for multiple emails in email field, employee imports updated documentation for /absencesf endpoint
2.7	08.12.2021	IS	15.3	Added /import/employeeandposition endpoint, minor correction to /absencesf endpoint
2.7.1	16.12.2021	IS	15.3	Fixed wrong endpoint names in 2.2.4.2 and 2.2.4.3
2.7.2	03.02.2022	RS	15.3	Added /exportToApi/UpdateRecordReceipts endpoint in 2.2.8
2.7.3	07.02.2022	RS	15.3	Added support for correlation Id in /absencessf endpoint in 2.2.4.5
2.7.4	07.02.2022	RS	15.3	Removed planned comments for /import/employeeandposition in 2.2.4.3 As it is now implemented
2.7.5	09.02.2022	RS	15.3	Added support for the “Deleted” field in position imports
2.8	04.03.2022	IS	15.3	Inserted chapter 5 on API based exports
2.8.1	05.05.2022	IS	16.0	Removed /export/schematimesf from swagger documentation Added WorkWeekHours
2.8.2	08.11.2022	IS	16.3	Added new functionality for handling employee import of Calendar and Wage group in 2.2.4.2 and 2.2.4.3
2.9	22.06.2023	HH	16.7	Added fields Phone for Employee and Hourly Wage for EmployeePosition
2.9.1	14.08.2023	IS	16.8	Added dynamic query for Daily Results in 2.2.8 (Export to Api moved to 2.2.9)
2.9.2	28.08.24	IS	17.2	Added support for apikeys in wintid api. Updated employee import optional texts.
2.9.3	14.10.24	IS	17.2.8	Added dynamic query for Employee (2.3.8.2), Job (2.3.8.3) and Work Environment Act (2.3.8.4)
2.9.4	13.12.24	IS HH	17.3.5	Added Employee import: snapshot by firm and included information about cutoffdate for snapshot import Added Responsible Use of API Endpoints
2.9.5	21.01.25	HH		Filled in the field descriptions for dynamic query Work Environment Act (2.3.8.4)
2.9.6	25.02.25	IS	17.3.13	Added description of Job import
2.9.7	11.03.25	IS	17.3.17	Added dynamic query for Registrations