DATA ACCESS CONCIERGE · GUIDE

Check that every Microsoft Entra access package ends access on a set date

A script that reads two files saved from Microsoft Graph and lists every Microsoft Entra access package policy and assignment that does not end access on a set date.

Published 2026-11-04 · Online version and downloads: www.datameaning.com/guides/check-every-entra-access-package-has-an-end-date/

Save your assignment policies and assignments from Microsoft Graph, then run the script. It lists every access package policy and assignment with no end date, a term longer than your maximum, a missed expiry, or an expiry it cannot read. It reads the files you save and changes nothing in Entra.

What you will need

- Python 3.9 or later and the `duckdb` module (tested on Python 3.12 and DuckDB 1.5.6).
- Permission to read entitlement management in Microsoft Graph: Microsoft lists `EntitlementManagement.Read.All` as the least privileged permission for the two calls below.
- The two saved answers (policies, and optionally assignments), each as one JSON file.
- Network access to download Microsoft's Graph metadata once per run (8,200,681 bytes; none of your data is sent), unless you pass `--no-schema-check`, in which case the result is INCOMPLETE.

What you will have at the end

A Python script, entra_end_date_check.py, that reads files you save from Microsoft Graph and lists every Microsoft Entra access package policy, and every assignment, that does not end access on a set date. It runs in DuckDB, reads the files you save and changes nothing in your tenant. Once per run it also downloads Microsoft's published Graph metadata (8,200,681 bytes) to check property names; none of your data is sent, and `--no-schema-check` skips the download, in which case the result is INCOMPLETE. Download the full guide (PDF) for the same steps with the complete code and both runs in one document.

Two words. A **policy** is the rule on an access package. Microsoft describes an access package assignment policy as specifying "the policy by which subjects can request or be assigned an access package", and its `expiration` property as "The expiration date for assignments created in this policy" (Microsoft Graph: accessPackageAssignmentPolicy (read 2026-11-04)). An **assignment** is a person's actual access: "an assignment of an access package to a particular subject and time range" (Microsoft Graph: accessPackageAssignment (read 2026-11-04)).

The script checks both, because a policy with an end date does not guarantee that each assignment has one. Microsoft's lifecycle page notes: "If you change the expiration date for assignments on a policy, the expiration date for requests that are already in a pending approval or approved state won't change" (Microsoft Learn: change lifecycle settings for an access package (read 2026-11-04)). A corrected policy can sit next to requests that kept the old date, which is one reason the script reads assignments as well. We did not test which other paths leave an assignment without an end date; the three assignment faults in Run 1 are planted to show that the check catches them.

The records in the two runs below are generated with a fixed seed, shaped as Graph returns them and not taken from any tenant. The Tested on box says so again.

Step 1. Save the two files from Microsoft Graph

Call the list of assignment policies, `GET /identityGovernance/entitlementManagement/assignmentPolicies` (Microsoft Graph: list assignmentPolicies (read 2026-11-04)), and the list of assignments, `GET /identityGovernance/entitlementManagement/assignments?$expand=accessPackage,assignmentPolicy` (Microsoft Graph: list accessPackageAssignments (read 2026-11-04)). Microsoft lists `EntitlementManagement.Read.All` as the least privileged permission for both. For a directory-wide administrator, Microsoft says the assignments list includes current and expired assignments. Save each answer as one JSON file.

The script reads the `value` list in each file. If an answer ends with an `@odata.nextLink`, the list continues on another page: the script does not guess, it reports the file as incomplete (Step 4). The script reads these properties and no others: on a policy `id`, `displayName`, `expiration` and `requestorSettings.allowCustomAssignmentSchedule`; on an assignment `id`, `state`, `schedule`, `accessPackage.displayName` and `assignmentPolicy.id`. It never reads the person an assignment belongs to.

Step 2. Decide what counts as an end date

Graph describes an expiration with an `expirationPattern`: a `type`, plus a `duration` or an `endDateTime` (Microsoft Graph: expirationPattern (read 2026-11-04)). The script turns each type into a result. `MAX_DAYS` is your own rule; Microsoft's lifecycle page lets an administrator set a number of days from 0 to 3660, or Never (Microsoft Learn: lifecycle settings (read 2026-11-04)).

How each expiration is judged

Type in Graph	What Microsoft says	Result in the script
`noExpiration`	"The requestor did not wish the access to expire."	NO END DATE
`notSpecified`	"No expiration schedule was specified."	NO END DATE
`afterDuration`	"Access will expire after a specified duration relative to access being granted."	TOO LONG if the duration is longer than `MAX_DAYS`; otherwise OK. A duration in months or years is not read: EXPIRATION NOT READ.
`afterDateTime`	"Access will expire after a specified date and time."	END DATE PASSED if on or before `AS_OF`; TOO LONG if later than `AS_OF` plus `MAX_DAYS`; otherwise OK.
anything else, or missing		EXPIRATION NOT READ. Never OK.

Microsoft's wording describes each type. The result label is the script's own judgment.

For an assignment the script counts only the ones giving access: any state except `expired` and `deliveryFailed`, including an assignment with no state (Microsoft Graph: accessPackageAssignment states (read 2026-11-04)). It works out the end from the schedule (Microsoft Graph: entitlementManagementSchedule (read 2026-11-04)) and judges the length from the start: no end date is NO END DATE, an end more than `MAX_DAYS` after the start is TOO LONG, and an end already past while the state is still not expired is PAST END DATE, NOT EXPIRED.

A policy that lets a requester propose their own timeline is not failed by that alone. Microsoft says `allowCustomAssignmentSchedule` set to false "indicates that the requestor isn't permitted to include a schedule in their request" (Microsoft Graph: accessPackageAssignmentRequestorSettings (read 2026-11-04)). The script lists the policies where it is true for review, and the assignment check shows what end dates those requests really got.

Step 3. Classify every policy and assignment in one query

The judgment is two `CASE` expressions in DuckDB, copied from the script. The order of the lines is the order of precedence.

Policies

```
SELECT *, CASE
WHEN etype IN ('noExpiration','notSpecified') THEN 'NO END DATE'
WHEN etype = 'afterDuration' AND dur_days IS NOT NULL AND dur_days > $2 THEN 'TOO LONG'
```

```
WHEN etype = 'afterDuration' AND dur_days IS NOT NULL THEN 'OK'
WHEN etype = 'afterDateTime' AND end_dt IS NOT NULL AND end_dt <= $1 THEN 'END DATE PASSED'
WHEN etype = 'afterDateTime' AND end_dt IS NOT NULL AND end_dt > $1 + to_days(CAST($2 AS INTEGER)) THEN 'TOO LONG'
WHEN etype = 'afterDateTime' AND end_dt IS NOT NULL THEN 'OK'
ELSE 'EXPIRATION NOT READ' END AS status FROM pol
```

Assignments

```
SELECT *, CASE
WHEN etype IN ('noExpiration','notSpecified') THEN 'NO END DATE'
WHEN ends_at IS NULL OR start_dt IS NULL THEN 'EXPIRATION NOT READ'
WHEN ends_at <= $1 THEN 'PAST END DATE, NOT EXPIRED'
WHEN date_diff('second', start_dt, ends_at) > $2 * 86400 THEN 'TOO LONG'
ELSE 'OK' END AS status FROM asg_result WHERE in_scope
```

Step 4. Refuse to say PASS on an input that may be incomplete

Before it judges anything, the script runs four input checks. If any fails, the result is INCOMPLETE, never PASS. Each one has a fault planted in Run 1 that must trip it.

Input checks

Check	What it catches	Planted fault that trips it
schema	A property name the script reads that is not in Microsoft's published Graph metadata, pinned to a commit and to a byte size and SHA-256 (the file, msgraph-metadata, MIT license (read 2026-11-04)).	A made-up property, `expiresOn`, added to the list the script reads
paging	A file whose answer carries an `@odata.nextLink`: part of the list is on another page.	A next link added to the policies file
ids	A policy with no id, or an id listed twice, which would count a policy twice.	The first policy listed a second time
references	An assignment that names no policy, or one that is not in the policies file, so its policy was not checked. Expired assignments and ones that failed to deliver are left out, as they are from the assignment check in Step 3.	An assignment naming a policy id that is not in the file

In a planted run the script also checks itself: it stops with "PLANTED FAULT NOT REPORTED" if any planted fault does not show up in the output, and it fails if a planted record could not be planted.

The code

The complete download, `entra_end_date_check.py`, at https://www.datameaning.com/guides/check-every-entra-access-package-has-an-end-date/entra_end_date_check.py:

```
#!/usr/bin/env python3
"""Check that every Microsoft Entra access package ends access on a set date.

Reads two files saved from Microsoft Graph (v1.0, Microsoft Entra entitlement management):
  * the assignment policies (GET /identityGovernance/entitlementManagement/assignmentPolicies), and
  * optionally the access package assignments (GET /identityGovernance/entitlementManagement/assignments),
and lists every policy, and every assignment, that has no end date, an end date farther away than your
maximum, an end date already passed but still active, or an expiration the script cannot read.

    python3 entra_end_date_check.py            # the generated demo records (clean)
    python3 entra_end_date_check.py --plant    # the same records with faults planted; the checks must catch them
    python3 entra_end_date_check.py --strict   # exit 1 unless the result is PASS (for a pipeline)

To run it on your own tenant, set POLICIES_FILE, ASSIGNMENTS_FILE, MAX_DAYS and AS_OF (the day you run it) below, nothing else.

Needs Python 3.9 or later and the duckdb module (tested on Python 3.12, DuckDB 1.5.6). The records of the
demo are generated with a fixed seed and are not from any tenant; every property name they use is checked
against Microsoft's published Graph metadata, pinned to a commit and downloaded once per run (8 MB; none of
your data is sent; --no-schema-check skips it and the result is then INCOMPLETE). The script writes no file and prints no
person: it reads policy names, access package names and ids only.
```

```
"""
import argparse
import datetime as dt
import hashlib
import json
import pathlib
import random
import re
import sys
import urllib.request
import uuid
import xml.etree.ElementTree as ET

import duckdb

# ---- Settings: the lines a reader changes ------------------------------------------------------
POLICIES_FILE = None      # path to the JSON saved from the assignmentPolicies call; None = generated demo
ASSIGNMENTS_FILE = None   # path to the JSON saved from the assignments call; None = demo (or "not checked")
MAX_DAYS = 90             # the longest access you allow, in days (90 is the demo's value, not a rule)
AS_OF = "2026-11-04"      # the day the check is run for, YYYY-MM-DD; set it to today
# ------------------------------------------------------------------------------------------------

SCHEMA_URL = ("https://raw.githubusercontent.com/microsoftgraph/msgraph-metadata/"
              "c4b98b7b7808ee5f4e7d752378339d70ef472154/clean_v10_metadata/cleanMetadataWithDescriptionsv1.0.xml")
SCHEMA_BYTES = 8_200_681
SCHEMA_SHA256 = "ff4f707d562dd1aa1dfaa591f4e17f4e6f50d517dc65b0278d216fb4fc4e547b"

# Every property this script reads, as (Graph type, property). Each must exist in the pinned schema.
READS = [
    ("entity", "id"),
    ("accessPackageAssignmentPolicy", "displayName"),
    ("accessPackageAssignmentPolicy", "expiration"),
    ("accessPackageAssignmentPolicy", "requestorSettings"),
    ("accessPackageAssignmentRequestorSettings", "allowCustomAssignmentSchedule"),
    ("expirationPattern", "type"),
    ("expirationPattern", "duration"),
    ("expirationPattern", "endDateTime"),
    ("accessPackageAssignment", "state"),
    ("accessPackageAssignment", "schedule"),
    ("accessPackageAssignment", "assignmentPolicy"),
    ("accessPackageAssignment", "accessPackage"),
    ("accessPackage", "displayName"),
    ("entitlementManagementSchedule", "startDateTime"),
    ("entitlementManagementSchedule", "expiration"),
]
# Enumerations whose values this script treats one by one; each value must exist in the pinned schema.
ENUMS = {
    "expirationPatternType": ["notSpecified", "noExpiration", "afterDateTime", "afterDuration"],
    "accessPackageAssignmentState": ["delivering", "partiallyDelivered", "delivered", "expired", "deliveryFailed"],
}
NO_END = ("noExpiration", "notSpecified")      # the expiration types that mean "never ends"
OUT_OF_SCOPE_STATES = ("expired", "deliveryFailed")   # an assignment in these states is not giving access

DURATION = re.compile(r"P(?:(\d+)W)?(?:(\d+)D)?(?:T(?:(\d+)H)?(?:(\d+)M)?(?:(\d+(?:\.\d+)?)S)?)?")
TIMESTAMP = re.compile(r"\d{4}-\d{2}-\d{2}T\d{2}:\d{2}:\d{2}(\.\d+)?(Z|[+-]\d{2}:\d{2})")


def parse_ts(v):
    """A Graph timestamp as a naive UTC datetime, or None if it is not one (never guessed)."""
    if not isinstance(v, str) or not TIMESTAMP.fullmatch(v.strip()):
        return None
    # exactly six decimals: Python before 3.11 reads only 3 or 6 in fromisoformat
    s = re.sub(r"\.(\d+)", lambda m: "." + (m.group(1) + "000000")[:6], v.strip())
    s = s[:-1] + "+00:00" if s.endswith("Z") else s
    try:
        return dt.datetime.fromisoformat(s).astimezone(dt.timezone.utc).replace(tzinfo=None)
    except ValueError:
        return None


def parse_days(v):
    """An ISO 8601 duration in weeks, days, hours, minutes, seconds as days. Years and months are not read."""
    if not isinstance(v, str):
        return None
    s = v.strip()
    m = DURATION.fullmatch(s)
```

```
    if not m or s == "P" or s.endswith("T"):
        return None
    w, d, h, mi, sec = (float(x) if x else 0.0 for x in m.groups())
    return w * 7 + d + h / 24 + mi / 1440 + sec / 86400


def check_schema(skip):
    """Fetch the pinned Graph metadata and check every name in READS and ENUMS. Returns (ok, message)."""
    if skip:
        return False, "schema: SKIPPED by --no-schema-check, so the property names were not checked"
    try:
        with urllib.request.urlopen(SCHEMA_URL, timeout=120) as resp:
            body = resp.read()
    except OSError as e:
        return False, f"schema: could not fetch the pinned Graph metadata ({e.__class__.__name__})"
    sha = hashlib.sha256(body).hexdigest()
    print(f"schema  {SCHEMA_URL.split('/')[-1]} at commit c4b98b7b, bytes={len(body)} sha256={sha[:16]}...")
    if len(body) != SCHEMA_BYTES or sha != SCHEMA_SHA256:
        return False, f"schema: the file is not the pinned one (expected {SCHEMA_BYTES} bytes, sha256 {SCHEMA_SHA256})"
    members, values = {}, {}
    for el in ET.fromstring(body).iter():
        tag = el.tag.rsplit("}", 1)[-1]
        if tag in ("EntityType", "ComplexType"):
            members[el.get("Name")] = {c.get("Name") for c in el if c.get("Name")}
        elif tag == "EnumType":
            values[el.get("Name")] = {c.get("Name") for c in el if c.get("Name")}
    missing = [f"{t}.{p}" for t, p in READS if p not in members.get(t, set())]
    missing += [f"{e}={v}" for e, vs in ENUMS.items() for v in vs if v not in values.get(e, set())]
    if missing:
        return False, "schema: not in the pinned Graph metadata: " + ", ".join(missing)
    return True, f"schema: all {len(READS)} properties and {sum(len(v) for v in ENUMS.values())} enumeration values the script
reads exist in the pinned metadata"


# ---- The demo records: generated with a fixed seed, shaped as Graph returns them ------------------------
def demo_records(as_of, plant):
    rng = random.Random(132)
    new_id = lambda: str(uuid.UUID(int=rng.getrandbits(128), version=4))
    midnight = lambda d: d.strftime("%Y-%m-%dT00:00:00Z")
    specs = [("Sales marts: read", "afterDuration", "P30D"), ("Marketing analytics: read", "afterDuration", "P60D"),
             ("Finance reporting: read", "afterDuration", "P90D"), ("Supply chain dashboards: read", "afterDuration", "PT1440H"),
             ("HR headcount views: read", "afterDuration", "P14D"), ("Customer 360 sandbox", "afterDateTime", "2026-12-
31T00:00:00Z"),
             ("Data science workspace", "afterDuration", "P45D"), ("Product usage tables: read", "afterDuration", "P90D")]

    def policy(name, kind, value, custom=False, pid=None):
        exp = {"type": kind}
        if kind == "afterDuration":
            exp["duration"] = value
        elif kind == "afterDateTime":
            exp["endDateTime"] = value
        return {"id": pid or new_id(), "displayName": name, "expiration": exp,
                "requestorSettings": {"allowCustomAssignmentSchedule": custom}}

    policies = [policy(*s) for s in specs]
    assignments, n = [], 0
    for p in policies:
        for _ in range(5):
            n += 1
            e = p["expiration"]
            if e["type"] == "afterDateTime":
                start = as_of - dt.timedelta(days=rng.randint(1, 30))
                end = parse_ts(e["endDateTime"])
            else:
                start = as_of - dt.timedelta(days=rng.randint(1, 80))
                end = start + dt.timedelta(days=parse_days(e["duration"]))
            state = "expired" if end <= as_of else "delivered"
            if n == 7:
                state = "deliveryFailed"
            exp = ({"type": "afterDateTime", "endDateTime": midnight(end)} if n % 2 else
                   {"type": "afterDuration", "duration": e.get("duration") or f"P{(end - start).days}D"})
            assignments.append({"id": new_id(), "state": state, "schedule": {"startDateTime": midnight(start), "expiration": exp},
                                "accessPackage": {"displayName": p["displayName"].replace(": read", "")},
                                "assignmentPolicy": {"id": p["id"]}})
    envelope = {"value": policies}
    expected = {"policy": {}, "assignment": {}, "input": set(), "custom": set()}
```

```
    if plant:
        pr = random.Random(1320)
        ids = {p["id"] for p in policies} | {a["id"] for a in assignments}
        fresh = lambda: str(uuid.UUID(int=pr.getrandbits(128), version=4))
        added = [("Finance close: read (quarter-end)", "noExpiration", None, False, "NO END DATE"),
                 ("Engineering logs: read", "afterDuration", "P365D", False, "TOO LONG"),
                 ("Partner portal tables", "afterDateTime", "2027-09-30T00:00:00Z", False, "TOO LONG"),
                 ("Legal matters: read", "afterDuration", "P3M", False, "EXPIRATION NOT READ"),
                 ("Research sandbox", "afterDuration", "P60D", True, "OK")]
        for name, kind, value, custom, want in added:
            p = policy(name, kind, value, custom, fresh())
            policies.append(p)
            expected["policy"][p["id"]] = want
            if custom:
                expected["custom"].add(p["id"])
        policies.append(dict(policies[0]))                      # the same policy listed twice
        expected["input"].add("ids")
        envelope["@odata.nextLink"] =
"https://graph.microsoft.com/v1.0/identityGovernance/entitlementManagement/assignmentPolicies?$skiptoken=demo"
        expected["input"].add("paging")
        good = policies[2]["id"]                                # "Finance reporting: read", a clean policy
        start = as_of - dt.timedelta(days=3)
        late = [({"startDateTime": midnight(start), "expiration": {"type": "noExpiration"}}, "delivered", good, "NO END DATE"),
                ({"startDateTime": midnight(start), "expiration": {"type": "afterDateTime", "endDateTime": midnight(start +
dt.timedelta(days=200))}},
                 "delivered", good, "TOO LONG"),
                ({"startDateTime": midnight(as_of - dt.timedelta(days=60)), "expiration": {"type": "afterDateTime", "endDateTime":
"2026-10-01T00:00:00Z"}},
                 "delivered", good, "PAST END DATE, NOT EXPIRED"),
                ({"startDateTime": midnight(start), "expiration": {"type": "afterDuration", "duration": "P30D"}}, "delivered",
fresh(), "OK")]
        for sched, state, pol, want in late:
            a = {"id": fresh(), "state": state, "schedule": sched, "accessPackage": {"displayName": "Finance reporting"},
                 "assignmentPolicy": {"id": pol}}
            assignments.append(a)
            expected["assignment"][a["id"]] = want
        expected["input"].add("references")
        READS.append(("accessPackageAssignmentPolicy", "expiresOn"))   # a name Graph does not publish
        expected["input"].add("schema")
        if not ids.isdisjoint({q["id"] for q in policies[8:13]} | set(expected["assignment"])):
            sys.exit("a planted record could not be planted: its id was already in the records")
    return envelope, {"value": assignments}, expected


def load_json(path):
    try:
        data = json.loads(pathlib.Path(path).read_text(encoding="utf-8"))
    except (OSError, ValueError) as e:
        sys.exit(f"cannot read {path} as JSON: {e.__class__.__name__}")
    return data


def records_of(data, label):
    """The list of records in a Graph answer: the 'value' list, or a bare list. Returns (records, next_link)."""
    if isinstance(data, list):
        return data, None
    if isinstance(data, dict) and isinstance(data.get("value"), list):
        return data["value"], data.get("@odata.nextLink")
    sys.exit(f"{label}: expected a JSON object with a 'value' list (as Graph returns) or a list of records")


def canon(v, names):
    """A string value written in Graph's own spelling, whatever case the export used; anything else as it is."""
    if not isinstance(v, str):
        return None
    return {n.lower(): n for n in names}.get(v.strip().lower(), v.strip())


def sub(d, key):
    v = d.get(key) if isinstance(d, dict) else None
    return v if isinstance(v, dict) else {}


def main():
    ap = argparse.ArgumentParser()
    ap.add_argument("--plant", action="store_true", help="plant faults in the generated demo records")
```

```
    ap.add_argument("--strict", action="store_true", help="exit 1 unless the result is PASS")
    ap.add_argument("--no-schema-check", action="store_true", help="do not fetch the pinned Graph metadata (the result is then
INCOMPLETE)")
    a = ap.parse_args()
    if not isinstance(MAX_DAYS, int) or isinstance(MAX_DAYS, bool) or MAX_DAYS < 1:
        sys.exit("MAX_DAYS must be a whole number of days, 1 or more")
    if not re.fullmatch(r"\d{4}-\d{2}-\d{2}", AS_OF):
        sys.exit("AS_OF must be written YYYY-MM-DD")
    as_of = dt.datetime.strptime(AS_OF, "%Y-%m-%d")
    demo = POLICIES_FILE is None
    if a.plant and not demo:
        sys.exit("--plant works only on the generated demo records (POLICIES_FILE must be None)")
    expected = {"policy": {}, "assignment": {}, "input": set(), "custom": set()}
    if demo:
        penv, aenv, expected = demo_records(as_of, a.plant)
        source = "generated demo records (seed 132), not from any tenant"
    else:
        penv = load_json(POLICIES_FILE)
        aenv = load_json(ASSIGNMENTS_FILE) if ASSIGNMENTS_FILE else None
        source = f"{POLICIES_FILE}" + (f" and {ASSIGNMENTS_FILE}" if ASSIGNMENTS_FILE else "")
    pols, p_next = records_of(penv, "policies")
    asg, a_next = records_of(aenv, "assignments") if aenv is not None else (None, None)

    print(f"check   every access package policy ends access within {MAX_DAYS} days; as of {AS_OF}")
    print(f"source  {source}")
    if not demo and AS_OF != dt.date.today().isoformat():
        print(f"note    AS_OF is {AS_OF}, not today ({dt.date.today().isoformat()}): both date checks compare with AS_OF")
    print(f"records {len(pols)} policies" + (f", {len(asg)} assignments" if asg is not None else ", assignments NOT CHECKED (no
ASSIGNMENTS_FILE)"))

    # --- input checks: a failed one makes the result INCOMPLETE, never PASS
    inputs = []   # (name, ok, text)
    ok, text = check_schema(a.no_schema_check)
    inputs.append(("schema", ok, text))
    nexts = [n for n in (p_next, a_next) if n]
    inputs.append(("paging", not nexts, "paging: every file is one whole answer" if not nexts else
                   f"paging: {len(nexts)} file(s) carry an @odata.nextLink, so the list continues on another page; join all pages
into one file and delete its @odata.nextLink"))

    con = duckdb.connect()
    con.execute("CREATE TABLE pol (row_no INTEGER, id VARCHAR, name VARCHAR, etype VARCHAR, dur_days DOUBLE, end_dt TIMESTAMP,
custom BOOLEAN)")
    for i, r in enumerate(pols, 1):
        r = r if isinstance(r, dict) else {}
        e, rs = sub(r, "expiration"), sub(r, "requestorSettings")
        pid = r.get("id") if isinstance(r.get("id"), str) and r.get("id").strip() else None
        cust = rs.get("allowCustomAssignmentSchedule")
        con.execute("INSERT INTO pol VALUES (?,?,?,?,?,?,?)",
                    [i, pid, r.get("displayName") if isinstance(r.get("displayName"), str) else "(no name)",
                     canon(e.get("type"), ENUMS["expirationPatternType"]), parse_days(e.get("duration")),
                     parse_ts(e.get("endDateTime")), cust if isinstance(cust, bool) else None])
    blank = con.execute("SELECT count(*) FROM pol WHERE id IS NULL").fetchone()[0]
    dup = con.execute("SELECT count(*) FROM (SELECT id FROM pol WHERE id IS NOT NULL GROUP BY id HAVING count(*) > 1)").fetchone()
[0]
    inputs.append(("ids", blank == 0 and dup == 0, "ids: every policy has its own id" if blank == 0 and dup == 0 else
                   f"ids: {blank} policies with no id and {dup} ids listed more than once (a repeated policy is counted twice)"))

    con.execute("CREATE TABLE asg (row_no INTEGER, id VARCHAR, package VARCHAR, policy_id VARCHAR, state VARCHAR, "
                "start_dt TIMESTAMP, etype VARCHAR, dur_days DOUBLE, end_dt TIMESTAMP)")
    for i, r in enumerate(asg or [], 1):
        r = r if isinstance(r, dict) else {}
        sch = sub(r, "schedule")
        e = sub(sch, "expiration")
        pid = sub(r, "assignmentPolicy").get("id")
        con.execute("INSERT INTO asg VALUES (?,?,?,?,?,?,?,?,?)",
                    [i, r.get("id") if isinstance(r.get("id"), str) else None,
                     sub(r, "accessPackage").get("displayName") if isinstance(sub(r, "accessPackage").get("displayName"), str) else
"(no name)",
                     pid if isinstance(pid, str) and pid.strip() else None,
                     canon(r.get("state"), ENUMS["accessPackageAssignmentState"]), parse_ts(sch.get("startDateTime")),
                     canon(e.get("type"), ENUMS["expirationPatternType"]), parse_days(e.get("duration")),
                     parse_ts(e.get("endDateTime"))])
    if asg is not None:
        # only assignments the script checks: an expired or failed one under a deleted policy is not an input fault
        orphans = con.execute("SELECT count(*) FROM asg a WHERE (a.state IS NULL OR a.state NOT IN ?) AND (a.policy_id IS NULL "
                              "OR NOT EXISTS (SELECT 1 FROM pol p WHERE p.id = a.policy_id))",
```

```
                            [list(OUT_OF_SCOPE_STATES)]).fetchone()[0]
        inputs.append(("references", orphans == 0, "references: every assignment names a policy that is in the policies file" if
orphans == 0 else
                       f"references: {orphans} assignments name no policy, or one that is not in the policies file, so their policy
was not checked"))

    print("\n== Input checks")
    for name, ok, text in inputs:
        print(f"  [{'ok' if ok else 'FAIL'}] {text}")

    # --- policies
    params = [as_of, MAX_DAYS]
    con.execute("""CREATE TABLE pol_result AS SELECT *, CASE
        WHEN etype IN ('noExpiration','notSpecified') THEN 'NO END DATE'
        WHEN etype = 'afterDuration' AND dur_days IS NOT NULL AND dur_days > $2 THEN 'TOO LONG'
        WHEN etype = 'afterDuration' AND dur_days IS NOT NULL THEN 'OK'
        WHEN etype = 'afterDateTime' AND end_dt IS NOT NULL AND end_dt <= $1 THEN 'END DATE PASSED'
        WHEN etype = 'afterDateTime' AND end_dt IS NOT NULL AND end_dt > $1 + to_days(CAST($2 AS INTEGER)) THEN 'TOO LONG'
        WHEN etype = 'afterDateTime' AND end_dt IS NOT NULL THEN 'OK'
        ELSE 'EXPIRATION NOT READ' END AS status FROM pol""", params)
    print(f"\n== Policies: does each one end access within {MAX_DAYS} days?")
    for st, n in con.execute("SELECT status, count(*) FROM pol_result GROUP BY status ORDER BY (status = 'OK') DESC,
status").fetchall():
        print(f"  {st:<22}{n:>4}")
    rows = con.execute("SELECT row_no, id, name, status, etype, dur_days, end_dt FROM pol_result WHERE status <> 'OK' ORDER BY
row_no").fetchall()
    for row_no, pid, name, st, etype, dur, end in rows:
        detail = (f"duration {dur:g} days" if etype == "afterDuration" and dur is not None else
                  f"ends {end:%Y-%m-%d}" if etype == "afterDateTime" and end is not None else
                  "duration missing or not in weeks, days, hours, minutes or seconds" if etype == "afterDuration" else
                  "end date missing or not a timestamp" if etype == "afterDateTime" else f"type {etype or 'missing'}")
        print(f"  - row {row_no:>2}  {st:<20} {name} ({(pid or 'no id')[:8]}): {detail}")
    custom = con.execute("SELECT row_no, id, name FROM pol_result WHERE custom ORDER BY row_no").fetchall()
    unknown = con.execute("SELECT count(*) FROM pol_result WHERE custom IS NULL").fetchone()[0]
    print(f"  requesters may propose their own timeline on {len(custom)} policies; setting not in the file for {unknown}")
    for row_no, pid, name in custom:
        print(f"  - row {row_no:>2}  REVIEW               {name} ({(pid or 'no id')[:8]}): look at the end dates its assignments
really got")

    # --- assignments
    a_bad = 0
    if asg is not None:
        con.execute("""CREATE TABLE asg_result AS SELECT *, state IS NULL OR state NOT IN ('expired','deliveryFailed') AS in_scope,
            CASE WHEN etype = 'afterDuration' AND start_dt IS NOT NULL AND dur_days IS NOT NULL THEN start_dt +
to_seconds(CAST(dur_days * 86400 AS BIGINT))
                 WHEN etype = 'afterDateTime' THEN end_dt END AS ends_at FROM asg""")
        con.execute("""CREATE TABLE asg_status AS SELECT *, CASE
            WHEN etype IN ('noExpiration','notSpecified') THEN 'NO END DATE'
            WHEN ends_at IS NULL OR start_dt IS NULL THEN 'EXPIRATION NOT READ'
            WHEN ends_at <= $1 THEN 'PAST END DATE, NOT EXPIRED'
            WHEN date_diff('second', start_dt, ends_at) > $2 * 86400 THEN 'TOO LONG'
            ELSE 'OK' END AS status FROM asg_result WHERE in_scope""", params)
        out_of_scope = con.execute("SELECT count(*) FROM asg_result WHERE NOT in_scope").fetchone()[0]
        print(f"\n== Assignments: does each one that is giving access end within {MAX_DAYS} days of starting?")
        print(f"  {out_of_scope} assignments are expired or failed to deliver and are not checked")
        for st, n in con.execute("SELECT status, count(*) FROM asg_status GROUP BY status ORDER BY (status = 'OK') DESC,
status").fetchall():
            print(f"  {st:<28}{n:>4}")
        for row_no, aid, pkg, pid, st, start, ends in con.execute(
                "SELECT row_no, id, package, policy_id, status, start_dt, ends_at FROM asg_status WHERE status <> 'OK' ORDER BY
row_no").fetchall():
            days = f", {(ends - start).total_seconds() / 86400:g} days" if ends is not None and start is not None else ""
            print(f"  - row {row_no:>2}  {st:<26} {pkg} (assignment {(aid or 'no id')[:8]}, policy {(pid or 'none')[:8]}){days}")
        a_bad = con.execute("SELECT count(*) FROM asg_status WHERE status <> 'OK'").fetchone()[0]

    p_bad = con.execute("SELECT count(*) FROM pol_result WHERE status <> 'OK'").fetchone()[0]
    failed = [n for n, ok, _ in inputs if not ok]
    result = "INCOMPLETE" if failed else "FAIL" if (p_bad or a_bad) else "PASS"
    print()
    if a.plant:
        # every planted fault must have been reported, or the check itself is broken
        wrong = []
        for pid, want in expected["policy"].items():
            got = con.execute("SELECT status FROM pol_result WHERE id = ? ORDER BY row_no LIMIT 1", [pid]).fetchone()
            if not got or got[0] != want:
                wrong.append(f"policy {pid[:8]} expected {want}, got {got[0] if got else 'nothing'}")
```

```
        for pid in expected["custom"]:
            if not con.execute("SELECT 1 FROM pol_result WHERE id = ? AND custom", [pid]).fetchone():
                wrong.append(f"policy {pid[:8]} expected REVIEW")
        for aid, want in expected["assignment"].items():
            got = con.execute("SELECT status FROM asg_status WHERE id = ?", [aid]).fetchone()
            if not got or got[0] != want:
                wrong.append(f"assignment {aid[:8]} expected {want}, got {got[0] if got else 'nothing'}")
        wrong += [f"input check {n} did not fail" for n in expected["input"] if n not in failed]
        if wrong:
            print("PLANTED FAULT NOT REPORTED: " + "; ".join(wrong))
            return 1
        print(f"PLANTED: {len(expected['policy'])} policies, {len(expected['assignment'])} assignments and "
              f"{len(expected['input'])} input faults; every one was reported")
    print(f"RESULT: {result}" + {"INCOMPLETE": f" (input checks failed: {', '.join(failed)}; "
                                               f"{p_bad} policies and {a_bad} assignments to fix as well)",
                                 "FAIL": f" ({p_bad} policies and {a_bad} assignments to fix)",
                                 "PASS": "" if asg is not None else " (policies only; assignments not checked)"}[result])
    return 1 if (a.strict and result != "PASS") else 0


if __name__ == "__main__":
    sys.exit(main())
```

The runs

Run 1: plant faults and see whether the script notices

We ran the script on 2026-11-04 with `--plant`, which adds faults to the generated records. It plants five policies (one with no end date, one of 365 days, one ending 2027-09-30, one with a duration in months, one that lets requesters propose a timeline), four assignments (one of them names a policy that is not in the file), and the other three input faults in Step 4: a made-up property, a next link and a repeated policy. Each row below says which line of the output reports it.

Command: run 1

```
python3 guides/check-every-entra-access-package-has-an-end-date/entra_end_date_check.py --plant
```

Output from our run 1

```
check   every access package policy ends access within 90 days; as of 2026-11-04
source  generated demo records (seed 132), not from any tenant
records 14 policies, 44 assignments
schema  cleanMetadataWithDescriptionsv1.0.xml at commit c4b98b7b, bytes=8200681 sha256=ff4f707d562dd1aa...

== Input checks
  [FAIL] schema: not in the pinned Graph metadata: accessPackageAssignmentPolicy.expiresOn
  [FAIL] paging: 1 file(s) carry an @odata.nextLink, so the list continues on another page; join all pages into one file and delete
its @odata.nextLink
  [FAIL] ids: 0 policies with no id and 1 ids listed more than once (a repeated policy is counted twice)
  [FAIL] references: 1 assignments name no policy, or one that is not in the policies file, so their policy was not checked

== Policies: does each one end access within 90 days?
  OK                     10
  EXPIRATION NOT READ     1
  NO END DATE             1
  TOO LONG                2
  - row  9  NO END DATE          Finance close: read (quarter-end) (22f2846e): type noExpiration
  - row 10  TOO LONG             Engineering logs: read (6d7fb6be): duration 365 days
  - row 11  TOO LONG             Partner portal tables (f4f3f9c2): ends 2027-09-30
  - row 12  EXPIRATION NOT READ  Legal matters: read (bde304ed): duration missing or not in weeks, days, hours, minutes or seconds
  requesters may propose their own timeline on 1 policies; setting not in the file for 0
  - row 13  REVIEW               Research sandbox (ff252dc5): look at the end dates its assignments really got

== Assignments: does each one that is giving access end within 90 days of starting?
  12 assignments are expired or failed to deliver and are not checked
  OK                          29
  NO END DATE                  1
  PAST END DATE, NOT EXPIRED   1
  TOO LONG                     1
  - row 41  NO END DATE               Finance reporting (assignment c93d04ba, policy c6ca90c2)
  - row 42  TOO LONG                  Finance reporting (assignment bd4f26c5, policy c6ca90c2), 200 days
```

```
  - row 43  PAST END DATE, NOT EXPIRED Finance reporting (assignment f808ab74, policy c6ca90c2), 26 days

PLANTED: 5 policies, 4 assignments and 4 input faults; every one was reported
RESULT: INCOMPLETE (input checks failed: schema, paging, ids, references; 4 policies and 3 assignments to fix as well)
```

What run 1 reported

Planted fault	Reported as
Policy with type `noExpiration`	NO END DATE, row 9
Policy with a duration of 365 days	TOO LONG, row 10
Policy ending 2027-09-30, farther than the maximum from the run date	TOO LONG, row 11
Policy with a duration in months	EXPIRATION NOT READ, row 12
Policy that lets requesters propose a timeline	REVIEW, row 13
Assignment under a good policy with no end date	NO END DATE, row 41
Assignment that runs 200 days	TOO LONG, row 42
Assignment past its end date, still delivered	PAST END DATE, NOT EXPIRED, row 43
Made-up property, next link, repeated policy, unknown policy id	Four [FAIL] input checks, so RESULT: INCOMPLETE

Two things to read in it. The three assignment faults sit under a policy that is itself OK: checking policies alone would have passed them. And the result is INCOMPLETE rather than FAIL because the input checks failed; the findings are still listed and counted in the last line.

Run 2: the clean records

The same script with no planted fault, run the same day. Every input check passes, every policy and every in-scope assignment has an end date within the maximum, and the result is PASS. PASS describes these files, not whether access ended. A policy listed for REVIEW does not change the result, so read those lines even on a PASS.

Command: run 2

```
python3 guides/check-every-entra-access-package-has-an-end-date/entra_end_date_check.py
```

Output from our run 2

```
check   every access package policy ends access within 90 days; as of 2026-11-04
source  generated demo records (seed 132), not from any tenant
records 8 policies, 40 assignments
schema  cleanMetadataWithDescriptionsv1.0.xml at commit c4b98b7b, bytes=8200681 sha256=ff4f707d562dd1aa...

== Input checks
  [ok] schema: all 15 properties and 9 enumeration values the script reads exist in the pinned metadata
  [ok] paging: every file is one whole answer
  [ok] ids: every policy has its own id
  [ok] references: every assignment names a policy that is in the policies file

== Policies: does each one end access within 90 days?
  OK                      8
  requesters may propose their own timeline on 0 policies; setting not in the file for 0

== Assignments: does each one that is giving access end within 90 days of starting?
  12 assignments are expired or failed to deliver and are not checked
  OK                          28

RESULT: PASS
```

The twelve assignments that are expired or failed to deliver are counted and left out, because they are not giving access. The output prints that count, so a reader can see how many rows were set aside. The script exits 0 by default; add `--strict` and it exits 1 unless the result is PASS, so a pipeline can stop on a missing end date.

Run it on your own tenant

To run it on your own tenant, save the answer of the assignment policies call as one JSON file, and, if you want assignments checked too, the answer of the assignments call (with $expand=accessPackage,assignmentPolicy) as another. If an answer ends with an @odata.nextLink, request that link as well and put the value lists of all the pages into the same file, then delete the @odata.nextLink from that file. In the download, change four lines near the top and nothing else: set POLICIES_FILE to the path of the policies file, ASSIGNMENTS_FILE to the path of the assignments file (leave it None to skip assignments), MAX_DAYS to the longest access you allow, in days, and AS_OF to the date of the day you run it, as YYYY-MM-DD.

Nothing else changes. The script prints policy names, access package names and the first eight characters of an id, never a person. A result of INCOMPLETE is not a pass: read which input check failed and fix the file. We also ran these steps on a made-up tenant file with different file names, a differently cased expiration type, a timestamp with seven decimals and an assignment that names no policy. The script read the file as intended and reported the missing policy as INCOMPLETE. The script still needs network access to fetch the metadata.

How to read the results and what a person does

Results and next steps

Result	What it means	What a person does
NO END DATE	Nothing sets an end date for this policy or assignment.	An access package manager sets an expiration on the policy, and decides what to do with the assignments already granted.
TOO LONG	The end is farther away than `MAX_DAYS`.	Shorten the policy, or record why this access needs longer.
END DATE PASSED	A policy's fixed end date is on or before `AS_OF`.	Set a new date, or retire the policy.
PAST END DATE, NOT EXPIRED	An assignment's end has passed but its state still says it is giving access.	Look at that assignment in Entra before trusting the end date.
EXPIRATION NOT READ	The file lacks the expiration, or holds a duration the script does not read.	Fix the export, or convert the months to days.
REVIEW	Requesters may propose their own timeline on this policy.	Look at the end dates its assignments really got, in the assignment table.
INCOMPLETE	An input check failed, so the lists may not be whole.	Fix the file the check names, then run again.

Checklist

Checklist

Done	Item
□	Saved the assignment policies answer as one JSON file, with every page joined
□	Saved the assignments answer with `$expand=accessPackage,assignmentPolicy`
□	Set `POLICIES_FILE`, `ASSIGNMENTS_FILE`, `MAX_DAYS` and `AS_OF`, and nothing else
□	All four input checks print [ok]
□	Every policy listed as NO END DATE, TOO LONG, END DATE PASSED or EXPIRATION NOT READ has an owner and a fix
□	Every assignment listed has been looked at in Entra
□	Every policy marked REVIEW has had its assignments' end dates read
□	Run again after the fixes and kept the PASS output

Method

- **Source of the records:** generated in the script with a fixed seed (132), shaped as Microsoft Graph v1.0 returns an assignment policy and an assignment. They are not from any tenant. A reader's files replace them.
- **Property names:** every property the script reads is checked against Microsoft's Graph metadata for v1.0 at a pinned commit, file size 8,200,681 bytes, with its SHA-256 printed in the run. The check also confirms every enumeration value the script treats one by one.
- **Judgment:** the two `CASE` expressions in Step 3, written out so a reader can disagree with them. `MAX_DAYS` and `AS_OF` are settings, not Microsoft rules.
- **Dates:** only timestamps written as full ISO 8601 with a time and a zone are read; anything else counts as not read, never guessed. Durations are read in weeks, days, hours, minutes and seconds.
- **Order:** the detail lines are printed in file order, so a re-run on the same files prints the same lines.
- **Self-checks, each with the planted fault that trips it:** the four input checks in Step 4, and, in a planted run, that each planted record is reported with the status planted for it.

Limits

- **This is not a client result and not a run on a tenant.** The records are generated. We shaped them as Microsoft's documentation describes Graph's answers, and we have not run the script against a live Entra tenant, so check the first run on your own files by reading the counts.
- **An end date in Entra is not removal from your data platforms.** The script reads Entra's own settings. It does not check that a group, a role or a grant in a warehouse really ends on that date.
- **Extensions are not checked.** The lifecycle tab can let users extend their access (Microsoft Learn (read 2026-11-04)). The script reads no extension setting, and we did not test whether an extension changes the schedule Graph returns for an assignment.
- **Requester-chosen timelines are listed, not judged.** We found no statement in Microsoft's documentation of what limit applies to an end date a requester proposes, so the script flags the policy and leaves the question to the assignment check.
- **A moment, not a history.** The files are what Graph returned when you saved them. Run it again after changes.
- **Names are printed.** Policy and access package names appear in the output. If those names are sensitive in your organization, treat the output accordingly.

When this is not enough

- **You need to know access really ended.** Compare the end dates with what your data platforms show for the same people. This script stops at Entra's settings.
- **You need the approval trail.** An end date says when access stops, not who agreed to it or why. That needs the request history, which is a separate record.
- **You have more than one tenant, or many policies and assignments.** Page through Graph, join the pages into one file per list, and run it once per tenant.
- **Finding a gap is not fixing it.** Someone with the right Entra role still has to change the policy and decide what happens to access already granted.

Sources

- Microsoft Graph: accessPackageAssignmentPolicy (read 2026-11-04): https://learn.microsoft.com/en-us/graph/api/resources/accesspackageassignmentpolicy?view=graph-rest-1.0

- Microsoft Graph: expirationPattern (read 2026-11-04): https://learn.microsoft.com/en-us/graph/api/resources/expirationpattern?view=graph-rest-1.0
- Microsoft Graph: accessPackageAssignmentRequestorSettings (read 2026-11-04): https://learn.microsoft.com/en-us/graph/api/resources/accesspackageassignmentrequestorsettings?view=graph-rest-1.0
- Microsoft Graph: accessPackageAssignment (read 2026-11-04): https://learn.microsoft.com/en-us/graph/api/resources/accesspackageassignment?view=graph-rest-1.0
- Microsoft Graph: entitlementManagementSchedule (read 2026-11-04): https://learn.microsoft.com/en-us/graph/api/resources/entitlementmanagementschedule?view=graph-rest-1.0
- Microsoft Graph: list assignmentPolicies (read 2026-11-04): https://learn.microsoft.com/en-us/graph/api/entitlementmanagement-list-assignmentpolicies?view=graph-rest-1.0
- Microsoft Graph: list accessPackageAssignments (read 2026-11-04): https://learn.microsoft.com/en-us/graph/api/entitlementmanagement-list-assignments?view=graph-rest-1.0
- Microsoft Learn: change lifecycle settings for an access package (read 2026-11-04): https://learn.microsoft.com/en-us/entra/id-governance/entitlement-management-access-package-lifecycle-policy
- Microsoft Graph v1.0 metadata (msgraph-metadata, pinned commit c4b98b7b, MIT license) (read 2026-11-04): https://github.com/microsoftgraph/msgraph-metadata/blob/c4b98b7b7808ee5f4e7d752378339d70ef472154/clean_v10_metadata/cleanMetadataWithDescriptionsv1.0.xml
- Data Access Concierge (Data Meaning) (read 2026-11-04): https://www.datameaning.com/solutions/data-access-concierge.html

Want access that ends on its own, without running a check?

This script is a check you run yourself. It is not part of a Data Meaning product, and it changes nothing in Entra.

Data Access Concierge is a separate service, not new software. We set it up in your own systems and run it for you. On Microsoft Entra, each request form (an access package) has an owner who approves, a required reason and an end date, and every permission ends on that date without anyone acting.

See how Data Access Concierge works: https://www.datameaning.com/solutions/data-access-concierge.html